Computational Methods for Hybrid Systems

Hedlund, Sven

1999

Document Version:
Publisher's PDF, also known as Version of record

Computational Methods for Hybrid Systems

Sven Hedlund

Department of Automatic Control
Lund Institute of Technology
Lund, September 1999

Department of Automatic Control
Lund Institute of Technology
Box 118
S-221 00 LUND
Sweden

ISSN 0280–5316
ISRN LUTFD2/TFRT--3225--SE

Printed in Sweden by Universitetstryckeriet,
Lund University, Lund 1999

Acknowledgments

This work has been supported by the EU/Esprit longtime research projects FAMIMO, Fuzzy Algorithms for Multiple Input Multiple Output systems and H^2C, Heterogeneous Hybrid Control.

I am very happy to work at the Department of Automatic Control in Lund, being surrounded by a lot of people that I admire for their brains as well as for their hearts. A special thanks to my supervisors, Anders Rantzer and Karl-Erik Årzén for always having time and being optimistic.

Having got the fortunate opportunity to study in various countries, I want to thank all of those who worked on their doctoral theses at Electrical Engineering, University of Newcastle, Australia during my master thesis — you made me first interested in further academic research. I also want to thank prof. Tetsuya Iwasaki at the Department of Control and Systems Engineering, Tokyo Institute of Technology, and the students of his lab for their great hospitality during my stay in Japan.

I thank you my mother and my brother, the most solid supports of my life. Thank you for putting up with spontaneous lectures about automatic control when my enthusiasm becomes too big. Gunnar Blomén, you inspire me as the connoisseur of the art of living. Thank you Jonas Svensson — you still live in my memory.

Introduction

This thesis mainly contains two different computational approaches for the analysis and synthesis of nonlinear systems. Each of these approaches is presented in a conference paper that describes parts of the theory and algorithms. MATLAB commands implementing the algorithms have been developed and the manuals for these are included herein as well.

The thesis consists of this introduction and four publications (cf. References on page 10):

Paper I S. Hedlund and M. Johansson "A Toolbox for Computational Analysis of Piecewise Linear Systems"

Manual I PWLTool — A Matlab Toolbox for Analysis of Piecewise Linear Systems

Paper II S. Hedlund and A. Rantzer "Optimal Control of Hybrid Systems"

Manual II A Matlab Tool for Optimal Control of Hybrid Systems

Motivation

Hybrid systems are systems that involve interaction between discrete and continuous dynamics. Such systems have been studied with grow-

ing interest and activity in recent years. One reason for the interest is that modeling and simulation of a complex system often require a combination of mathematical models from a variety of engineering disciplines. The structure of such submodels can be very different, some can be discrete and some continuous.

Very often, the same phenomenon can be described either by a discrete model or a continuous one, depending on the context and purpose of the model [Antsaklis and Nerode, 1998]. Consider for example an indoor asynchronous discrete-event driven thermostat, which discretizes temperature information as {too cold, normal, too hot}. The temperature of the room is translated into these representations in the thermostat and the thermostat's response is translated back to electrical currents which control the furnace, air conditioner, blower, etc.

There are several reasons for using hybrid models to represent the dynamic behavior of interest. Reduction of complexity is accomplished in hybrid systems by incorporating models of dynamic processes at different levels of abstraction; for example, the thermostat in the above example sees a very simple model of the complex heat flow dynamics. Another example of complexity reduction is the approximation of a nonlinear system. The system can be split into a number of simpler systems (e.g. linear) that are switched between. This rather common approach in modeling physical phenomena is used in Paper I.

The frameworks that are used to study hybrid systems can be broadly divided into two categories: One category contains those systems that extend the traditional event-driven models to include continuously evolving variables, such as hybrid automata from computer science. The other category is based on traditional continuous systems that are extended to include discrete dynamics, such as the systems used in Paper II.

The need for methods of controlling hybrid systems is motivated by a vast number of examples, such as vehicle transmission systems, computer disk drives, constrained robotic systems, flexible manufacturing systems, sampled-data systems, intelligent highway systems, air traffic management systems, and various systems with relays, switches, and hysteresis.

Further general information about hybrid systems and related issues is found in the references of the papers of this thesis.

Model Specification

This thesis presents computational methods for two different kinds of systems. The model that is used in the first approach is a piecewise linear system that can be represented as

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

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.

The model (1) can be regarded as a hybrid system: the cell index may be viewed as a discrete state variable whose value changes when the continuous state hits a cell boundary. The discrete state plays, however, a very passive role in this case and the methods presented in Paper I and Manual I should rather be considered as tools for nonlinear systems without discrete modes. A more applicable hybrid extension of this model is discussed in [Johansson, 1999].

In this thesis the piecewise linear systems undergo various kinds of analysis based on piecewise quadratic Lyapunov functions. The analysis includes stability tests, L_2-gain, and output energy estimation. The tools also admit synthesis based on optimal control using piecewise quadratic cost functions: there are means of computing bounds on the value function and extracting the corresponding feedback control law.

In addition to being well suited for the analysis of piecewise linear systems such as linear systems with actuator limitations or gain-scheduled systems, this approach also could be used for approximation of nonlinear functions. One difficulty that has to be dealt with in the case of nonlinear approximation, however, is how to choose the polyhedral cell partition.

The model that is used in the second approach is a nonlinear hybrid system:

$$\begin{cases} \dot{x}(t) &= f_{q(t)}(x(t), u(t)) \\ q(t) &= \nu(x(t), q(t^-), \mu(t)) \end{cases} \tag{2}$$

where $x(t) \in X \subset \mathbf{R}^n$ is the state vector, $u(t) \in \Omega_u \subset \mathbf{R}^m$ is a continuous input signal of the system. There is also a discrete input,

$\mu(t) \in \Omega_\mu$, which allows for the selection between N different system modes, $q(t) \in Q = \{1, 2, \ldots, N\}$. The notation $q(t^-)$ is used for the left-hand limit of q at t. $S_{q,r}$ is a set (parameterized by q and r) such that switching from mode q to r is possible when $x \in S_{q,r} \subseteq X$.

The main issue that is addressed for this kind of systems is to find a value function that gives the minimum of the cost

$$J(x_0, q_0) = \int_{t_0}^{t_f} l_q(x, u)dt + \sum_{k=1}^{M} s(x(t_k), q(t_k^-), q(t_k^+)) \tag{3}$$

subject to (2) while bringing the system from an initial state (x_0, q_0) at time t_0, to a final state (x_f, q_f) at time t_f, where the end time, t_f, is free. Here, M is an arbitrary finite number of switches occurring at times $t_0 < t_1 < t_2 < \ldots < t_M < t_f$ and $s(x, q, r) > 0$ is an associated cost for switching from discrete state q to r, the continuous part being x just before the switch.

Though the value function also could be used for assessing stability for the system (by using e.g. the time to reach the final state as cost function), the focus is on control synthesis.

The main advantage of approach II is that it can handle a large class of nonlinear hybrid systems. Even for this approach however, there are difficulties involved in the choice of the statespace being considered: one of the main goals is to estimate the optimal cost for bringing the system from an initial state, (x_0, q_0) to a final state, (x_f, q_f), given that the continuous part of the state trajectory, $x(t)$, remains in X. Since the algorithm for solving this problem searches for a function that meet a set of inequalities in a mesh of points in X, it is desirable to keep X as small as possible. Many control problems do not experience state constraints that are of significant importance, leading to $X = \mathbf{R}^n$. The challenge is then to approximate X by a small region that still encloses the optimal trajectory for the original problem.

Both of the approaches above are currently limited in the size and complexity of the problems that can be addressed. Dealing with systems that have more than three continuous states could be painful concerning the computational time. There exist however ideas for further investigation on how to improve the speed.

Future work

Future work will be directed towards the hybrid models of the second approach in this thesis. Many interesting issues related to the optimal control of hybrid systems remain to be investigated. Some possible research extensions are listed below.

- *Refinement of the grid around an optimal trajectory.* As mentioned above, the complexity makes it difficult to compute the value function in a space with many states. If computing the value function in a single point rather than in a large subset of $X \times Q$, it should be sufficient to compute the value function along the optimal trajectory, keeping the problem at a reasonable size. Since the optimal trajectory is not known on beforehand, it is desirable to have an algorithm that starts using a sparse grid in a large region and then gradually refine the solution around the optimal trajectory.

- *"Clever" discretization of u.* The original system (2) is continuous in u. To get a numerical solution of the optimal control problem, u has to be discretized. The problem thus becomes reformulated to optimal control under a finite (small) number of discrete input signals. How should the discretization be chosen to render a value function that is close to the value function of the original problem?

- *Finding an upper bound of the value function.* Integration of the cost function during simulation does not provide an accurate estimate of the cost, since the granularity of the state space mesh makes it hard to decide when the final point is reached. Is there a better way of finding a relevant true upper bound?

- *Compare the LP formulation with fixed point iterations.* In Paper II, the value function of an optimal control problem is computed by reformulating a Bellman type inequality to a linear programming problem. In [Bardi and Capuzzo-Dolcetta, 1997], it is shown how to solve the Bellman equation of a purely continuous problem by means of fixed point iteration. This method should apply to hybrid systems as well, and a comparison between the

LP formulation and the fixed point iteration formulation should be made.

- *Testing the methods on real applications.* One of the industrial partners of the H^2C consortium is Daimler-Benz. The methods above will be tested on problems provided by this partner.

References

Antsaklis, P. J. and A. Nerode (1998): "Hybrid control systems: An introductory discussion to the special issue." *IEEE Transactions on Automatic Control*, **43:4**, pp. 457–460. Special issue on hybrid systems.

Bardi, M. and I. Capuzzo-Dolcetta (1997): *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations*. Birkhauser Boston.

Hedlund, S. and M. Johansson (1999a): "PWLTool — A Matlab toolbox for analysis of piecewise linear systems." Report TFRT-7582. Department of Automatic Control, Lund Institute of Technology, Lund,Sweden.

Hedlund, S. and M. Johansson (1999b): "A toolbox for computational analysis of piecewise linear systems." In *Proceedings of European Control Conference*. Karlsruhe.

Hedlund, S. and A. Rantzer (1999): "Optimal control of hybrid systems." In *IEEE Conference on Decision and Control*. Phoenix.

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

Paper I

A Toolbox for Computational Analysis of Piecewise Linear Systems

Sven Hedlund and Mikael Johansson

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. [Sontag, 1996]). Moreover, many popular control schemes, such as gain scheduling and fuzzy logic controllers, can be well modeled by piecewise linear systems (cf. [Årzén *et al.*, 1998]).

Recently, it has been shown how stability and performance of piecewise linear systems can be assessed using Lyapunov functions that are piecewise quadratic [Johansson and Rantzer, 1996]. 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[Corless, 1994], 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 [Hedlund and Johansson, 1999].

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 [Johansson, 1999]. This is e.g. useful for the analysis of fuzzy Takagi-Sugeno systems.

For convenient notation, we introduce

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

Table 1. Commands for building a PWL system.

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

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 nonnegative. We recognize this as the halfspace 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

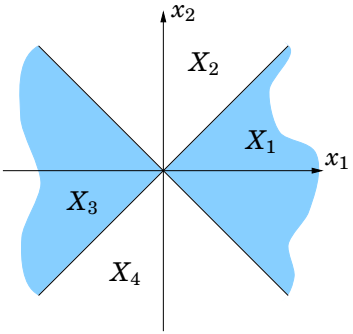

Figure 1. Partitions of the flower example.

$\bar{F}_i$-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 [Johansson and Rantzer, 1996] 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;
```

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 [Johansson, 1999] 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.

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

Table 2. Commands for defining hyperplane partitions.

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

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 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 [Johansson, 1999].

We illustrate the commands on a simple relay feedback system.

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

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

Table 3. Commands for defining simplex partitions.

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

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);
```

□

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. [Johansson, 1999]). 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 linear systems, problems may

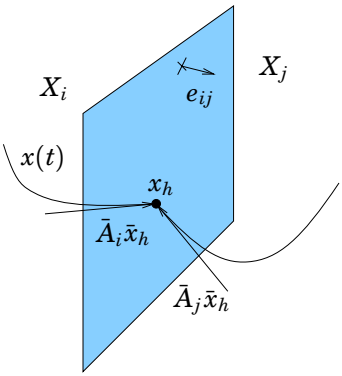

Figure 2. Sliding mode surface.

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 [Filippov, 1988; Utkin, 1977]. 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

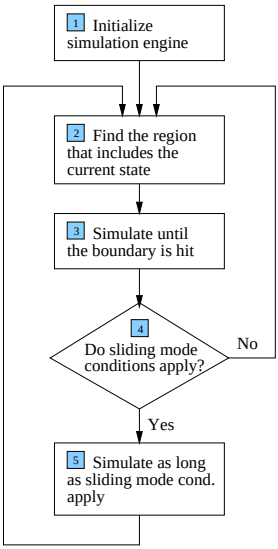

Figure 3. Schematical description of simulation algorithm for systems with sliding modes.

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 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

Table 4. Simulation related commands.

command	description
findsm	detect sliding modes
pwlsim	simulate PWL system

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 [Filippov, 1988]:

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

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 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

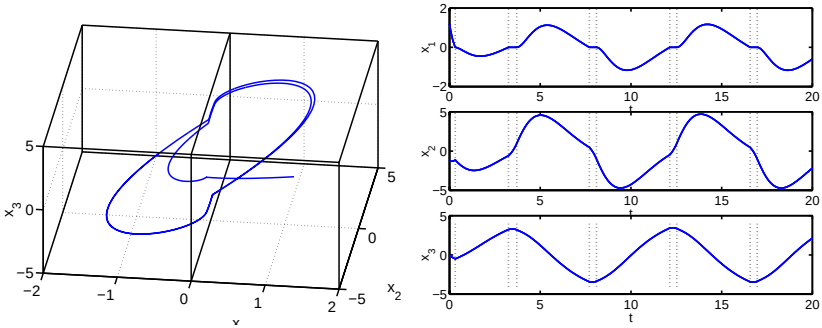

Figure 4. Limit cycle with sliding trajectory. The vertical dashed lines in the right part indicate time instances for the mode selection.

following nonminimum phase system from [Johansson, 1997]:

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

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. □

Table 5. Commands for stability analysis.

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

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 [Johansson, 1999].

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.

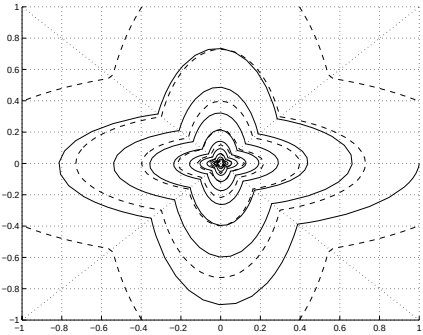

Figure 5. Simulation of the flower system (solid line) and level curves of a pwq lyapunov function (dashed).

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. □

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, $x(0)$.

Table 6. Commands for performance analysis and control design.

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

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 [Hedlund and Johansson, 1999] and additional examples.

Acknowledgements

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

8. References

Årzén, K.-E., M. Johansson, and R. Babuska (1998): "A survey on fuzzy control." Technical Report. Esprit LTR project FAMIMO Deliverable D1.1B.

Corless, M. (1994): "Robust stability analysis and controller design with quadratic Lyapunov functions." In Zinober, Ed., *Variable Structure and Lyapunov Control*, Lecture notes in Control and Information Sciences, chapter 9, pp. 181–203. Springer Verlag.

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

Hedlund, S. and M. Johansson (1999): "PWLTool — A Matlab toolbox for analysis of piecewise linear systems." Report TFRT-7582. Department of Automatic Control, Lund Institute of Technology, Lund,Sweden.

Johansson, K. H. (1997): *Relay Feedback and Multivariable Control*. PhD thesis ISRN LUTFD2/TFRT--1048--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Johansson, M. (1999): *Piecewise linear control systems*. PhD thesis ISRN LUTFD2/TFRT--1052--SE, Lund institute of technology.

Johansson, M. and A. Rantzer (1996): "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.

Sontag, E. (1996): "Interconnected automata and linear systems: A theoretical framework in discrete time." In Alur *et al.*, Eds., *Hybrid Systems III: Verification and Control*, pp. 436–448. Springer, NY.

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

Manual I

PWLTool
A Matlab Toolbox for Analysis of Piecewise Linear Systems

Sven Hedlund and Mikael Johansson

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 [Hedlund and Johansson, 1999]. The analysis and design are based on computation of piecewise quadratic Lyapunov functions [Rantzer and Johansson, 1997a]. The computations are performed using convex optimization in terms of linear matrix inequalities (LMIs). This version of the toolbox requires the LMI control toolbox [Gahine *et al.*, 1995].

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.

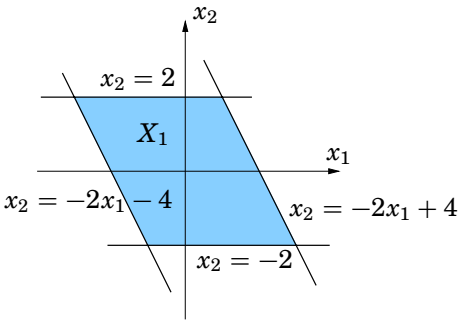

Figure 1. Example of a polyhedron in $\mathbf{R}^2$.

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 7 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 [Johansson and Rantzer, 1997]. This is e.g. useful for the analysis of fuzzy Takagi-Sugeno systems [Takagi and Sugeno, 1985].

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 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 [Rantzer and Johansson, 1997a].

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]

[1] The computations in [Johansson and Rantzer, 1996; Rantzer and Johansson, 1997b] 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.

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 [Johansson, 1999]. For the inexperienced user, who might find it difficult to create appropriate F-matrices, Section 3 presents means to overcome this problem.

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.

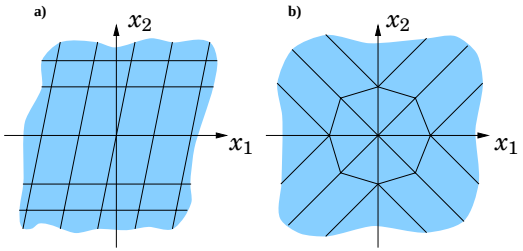

Figure 2. The special structures that are supported in sPWL package: **a)** Hyperrectangle partitions, **b)** Simplex partitions

4. Command Reference

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

1 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.

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 commands that are specific to a certain structure.

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).

2 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

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

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 [Gahine *et al.*, 1995]:

- `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 $x = (x_1, \ldots, x_N)$

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

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

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.

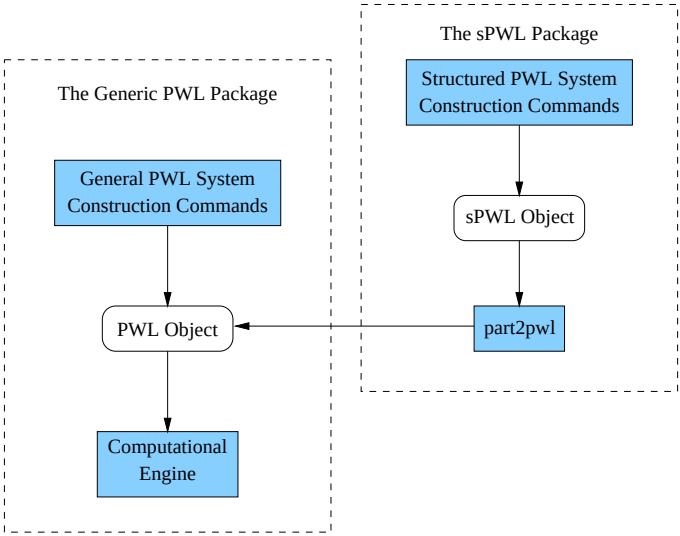

Figure 3. How the extra package for generating sPWL systems relates to the rest of the toolbox

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

$c^T x$ 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. Default value is `options(5) = 0`.

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, 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 for 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

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

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 the dynamics 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 a generic PWL command.

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 2).

`iogain` returns `gamma` $= \gamma$. `P` is a matrix resulting from the LMI calculations (as outlined in [Rantzer and Johansson, 1997a]). *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 [Rantzer and Johansson, 1997a]). *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 vector of control parameters (cf. Sec. 2).

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 56. The function `optcstub` also returns `O` that is a matrix resulting from the LMI calculations (as outlined in [Rantzer and Johansson, 1997a]). *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 2).

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 a generic PWL command.

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 2).

`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 [Rantzer and Johansson, 1997a]). *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 2).

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 2).

`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 [Hedlund and Johansson, 1999].

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` is 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 2).

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

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
>> A2 = [-0.1, 1; -5, -0.1];   % the 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);

>> 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
>> plot(xv(:,1), xv(:,2));                     % phase plane
>> 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

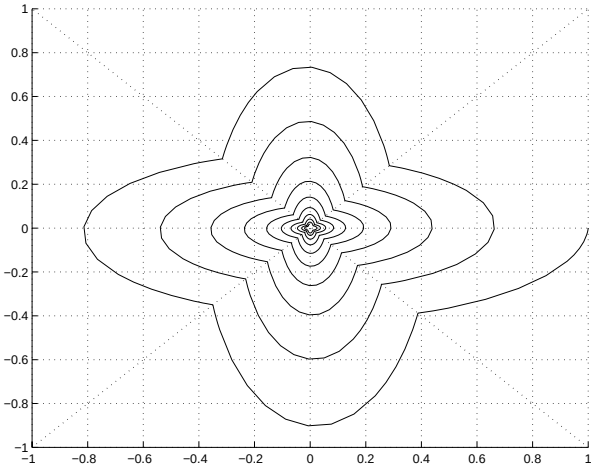

Figure 4. Simulation and cell partition of flower system

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.

```
>> 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

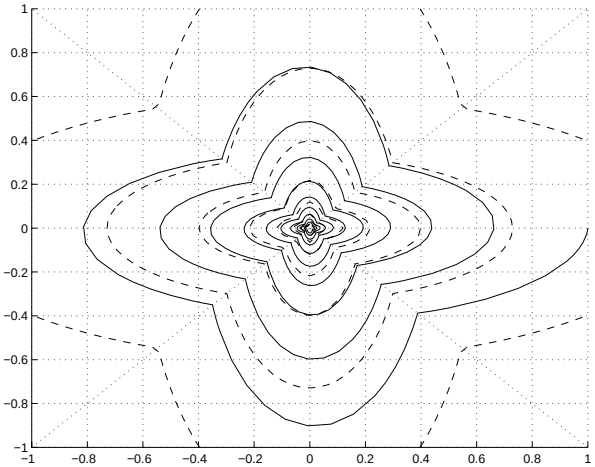

Figure 5. Level surfaces of the Lyapunov function

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. [Rantzer and Johansson, 1997a].

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 $x(0) = (1\ 2\ 3)'$.

```
>> 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.

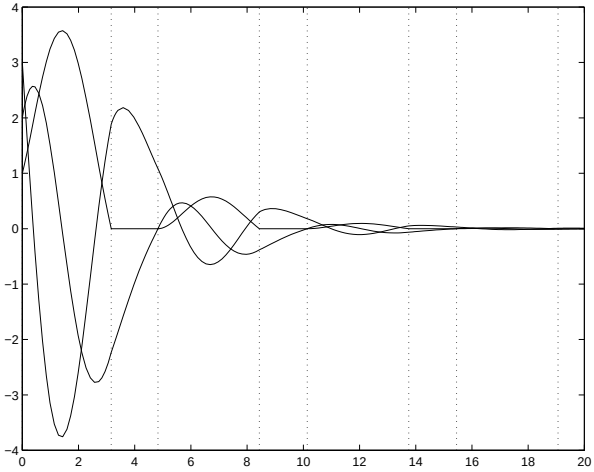

Figure 6. Trajectories from sliding mode simulation

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

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

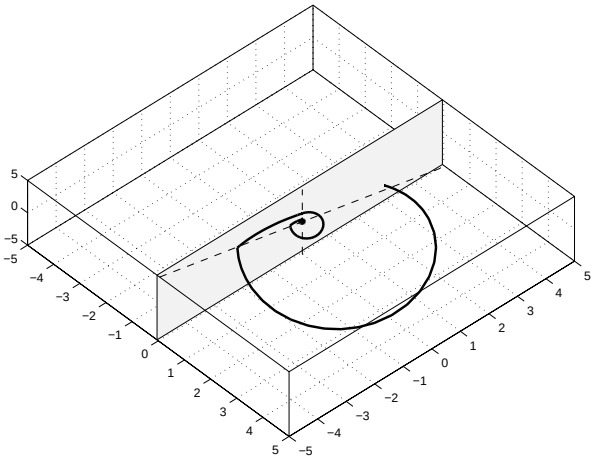

Figure 7. Three dimensional plot from sliding mode simulation

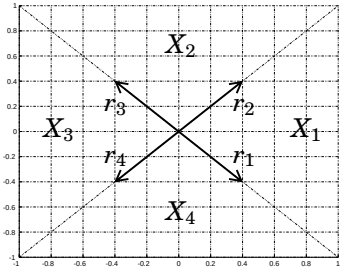

Figure 8. Simplex partitioning of the flower example.

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.)

```
>> 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
>> addscell([v1],[r2 r3],d2);       % to regions
>> 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
```

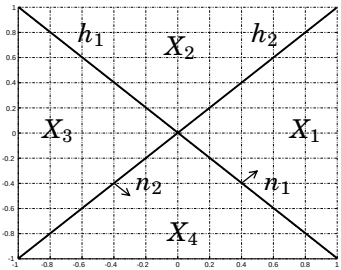

Figure 9. Hyperplane partitioning of the flower example.

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

Gahine, P., A. Nemirovski, A. Laub, and M. Chilali (1995): *LMI Control toolbox user's guide*. The MathWorks, inc.

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

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

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

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

Rantzer, A. and M. Johansson (1997a): "Piecewise linear quadratic optimal control." In *Proceedings of American Control Conference*. Albuquerque. Submitted for journal publication.

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

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

7. 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.

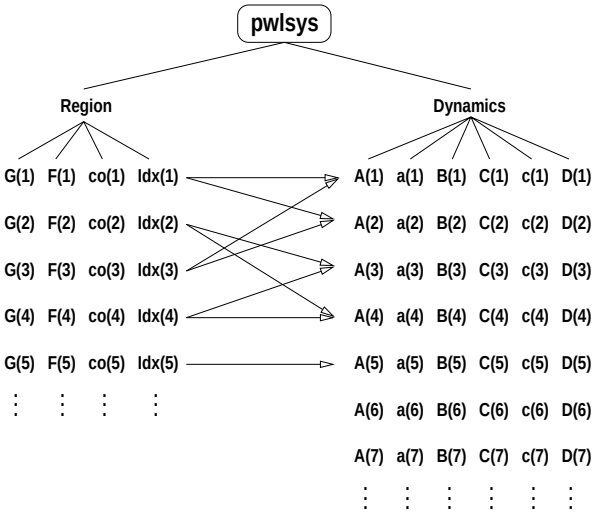

Figure 10. Schematic view of the data structure used for representing a PWL system in MATLAB

Paper II

Optimal Control of Hybrid Systems

Sven Hedlund and Anders Rantzer

Abstract

This paper presents a method for optimal control of hybrid systems. An inequality of Bellman type is considered and every solution to this inequality gives a lower bound on the optimal value function. A discretization of this "hybrid Bellman inequality" leads to a convex optimization problem in terms of finite-dimensional linear programming. From the solution of the discretized problem, a value function that preserves the lower bound property can be constructed. An approximation of the optimal feedback control law is given and tried on some examples.

1. Introduction

Hybrid systems are systems that involve interaction between discrete and continuous dynamics. Such systems have been studied with growing interest and activity in recent years. One reason for the interest is that modeling and simulation of a complex system often require a combination of mathematical models from a variety of engineering disciplines. The structure of such submodels can be very different, some can be discrete and some continuous.

Very often, the same phenomenon can be described either by a discrete model or a continuous one, depending on the context and purpose of the model [Antsaklis and Nerode, 1998]. Consider for example an

asynchronous discrete-event driven thermostat, which discretizes temperature information as {too hot, too cold, normal}.

Practical control systems typically involve switching between several different modes, depending on the range of operation. Even if the dynamics in each mode is simple and well understood, it is well known that automatic mode switching can give rise to unexpected phenomena.

Basic aspects of hybrid systems were treated in [Ezzine and Haddad, 1989], [Grossman *et al.*, 1993], and [Utkin, 1977]. For stability analysis, see [Branicky, 1998; Johansson, 1999] and references therein. The reformulation of an optimal control problem in terms of linear programming has previously been used for continuous time systems in [Rantzer, 1999] and [Rantzer and Johansson, 1997] and is closely connected to ideas of [Vinter and Lewis, 1978]. Related methods were discussed for discrete systems in [Bertsekas and Tsitsiklis, 1996] and on an abstract level for hybrid systems in [Branicky and Mitter, 1995].

This paper presents a novel computational approach to optimal control of hybrid systems, based on ideas from dynamic programming and convex optimization. Discretization of Bellman's inequality gives a lower bound on the optimal cost in terms of linear programming. A control law which is used for simulation is constructed from the lower bound. The results are demonstrated in some examples.

2. Problem Formulation

Define a hybrid system as

$$\begin{cases} \dot{x}(t) & = f_{q(t)}(x(t), u(t)) \\ q(t) & = v(x(t), q(t^-), \mu(t)) \end{cases} \tag{1}$$

where $x(t) \in X \subset \mathbf{R}^n$ is the state vector, $u(t) \in \Omega_u \subset \mathbf{R}^m$ is a continuous input signal of the system. There is also a discrete input, $\mu(t) \in \Omega_\mu$, which allows for the selection between N different system modes, $q(t) \in Q = \{1, 2, \ldots, N\}$. The notation $q(t^-)$ is used for the left-hand limit of q at t. $S_{q,r}$ is a set (parameterized by q and r) such that switching from mode q to r is possible when $x \in S_{q,r} \subseteq X$. The time argument, t, will often be omitted in the sequel for readability.

The optimal control problem is to minimize the cost function

$$J(x_0, q_0) = \int_{t_0}^{t_f} l_q(x,u)dt + \sum_{k=1}^{M} s(x(t_k), q(t_k^-), q(t_k^+)) \tag{2}$$

subject to (1) while bringing the system from an initial state (x_0, q_0) at time t_0, to a final state (x_f, q_f) at time t_f, where the end time, t_f, is free. Here, M is an arbitrary finite number of switches occurring at times $t_0 < t_1 < t_2 < \ldots < t_M < t_f$ and $s(x,q,r) > 0$ is an associated cost for switching from discrete state q to r, the continuous part being x just before the switch. Note that $s(\cdot) > 0$ removes the problem of infinitely many jumps in a finite interval.

The framework developed in this paper would also allow the number of continuous states to vary with the discrete mode according to $\dot{x}_q(t) = f_{q(t)}(x_q(t), u_q(t))$, where $x_q(t) \in X_q \subset \mathbf{R}^{n(q)}$, $u_q(t) \in \Omega_{u_q} \subset \mathbf{R}^{m(q)}$. The usage of the system description (1), however, will hopefully prevent the reader from getting stuck on details.

3. Lower Bounds on Optimal Cost

PROPOSITION 1
Let $V_q : X \mapsto R$, $q = 1, 2, \ldots, N$ be a set of continuous, piecewise C^1 functions that satisfy

$$0 \leq \frac{\partial V_q(x)}{\partial x} f_q(x,u) + l_q(x,u) \qquad \forall x \in X,\ u \in \Omega_u,\ q \in Q \tag{3}$$

$$0 \leq V_r(x) - V_q(x) + s(x,q,r) \qquad \forall x \in S_{q,r}\ \ q, r \in Q : q \neq r \tag{4}$$

$$0 = V_{q_f}(x_f) \tag{5}$$

where $f_q(x,u)$ gives the dynamics of a hybrid system according to (1), $l_q(x,u)$ and $s(x,q,r)$ define a cost function for the system according to (2). Then, for every (x_0, q_0), $V_{q_0}(x_0)$ gives a lower bound on the cost for

optimally bringing the system from (x_0, q_0) to (x_f, q_f), $x(t) \in X \ \forall t \in [t_0, t_f]$. □

Remark 1. Rather than having one single value function, $V(x)$, as would be the case for a purely continuous system, the proposition gives a set of value functions, $V_q(x)$, where q is the initial value of the discrete mode. Note that these functions give the cost for optimal trajectories that are allowed to switch modes — the index q only implies that trajectories *starting* in mode q are considered.

It is of course possible to think of $V_q(x)$ as one single function, parameterized by x and q. For consistent notation, however, $V_q(x)$ has been chosen instead of $V(x, q)$.

Proof. Let $\hat{u}(\cdot)$ and $\hat{\mu}(\cdot)$ be control signals that drive the system from (x_0, q_0) at time t_0 to (x_f, q_f) at time $t_f \equiv t_{M+1}$. Let $\hat{q}(t)$ denote the mode trajectory resulting from $\hat{\mu}(t)$ and define $x_k = x(t_k)$, $x_k^- = x(t_k^-)$, and $\hat{q}_k = \hat{q}(t)$, $t_k \leq t < t_{k+1}$. Then

$$
\begin{aligned}
J(x_0, \hat{q}_0) = & \\
& \sum_{k=0}^{M} \int_{t_k}^{t_{k+1}} l_{\hat{q}_k}(x, \hat{u}) dt + \sum_{k=1}^{M} s(x_k^-, \hat{q}_{k-1}, \hat{q}_k) \geq \\
& \sum_{k=0}^{M} \int_{t_k}^{t_{k+1}} -\frac{\partial V_{\hat{q}_k}(x)}{\partial x} f_{\hat{q}_k}(x, \hat{u}) dt + \\
& + \sum_{k=1}^{M} \left\{ V_{\hat{q}_{k-1}}(x_k^-) - V_{\hat{q}_k}(x_k^-) \right\} = \\
& \sum_{k=0}^{M} \left\{ V_{\hat{q}_k}(x_k) - V_{\hat{q}_k}(x_{k+1}) \right\} + \\
& + \sum_{k=1}^{M} \left\{ V_{\hat{q}_{k-1}}(x_k) - V_{\hat{q}_k}(x_k) \right\} = \\
& V_{\hat{q}_0}(x_0) - V_{\hat{q}_M}(x_{M+1}) = V_{\hat{q}_0}(x_0)
\end{aligned}
$$

□

Also the optimal value function, $V_q^\star(x)$ will meet the the constraints

(3)-(5), under appropriate interpretation of $\partial V_q(x)/\partial x$. Hence the inequalities do not introduce any conservatism in the lower bound.

4. Discretization

Utilizing a computer to solve (3)-(5) for a specific control problem, a straight forward approach is to grid the state space to require the inequalities to be met at a set of evenly distributed points in X. This approximation will, however, not guarantee a lower bound on the optimal cost, unless the nature of f_q and V_q *between* the grid points is taken into consideration.

In the case of a two-dimensional continuous state space, introduce the notation

$$
\begin{aligned}
x_{jk} &= x_f + jhe_1 + khe_2 \\
X^{jk} &= \{x_{jk} + \theta_1 he_1 + \theta_2 he_2 : \ 0 \le \theta_i \le 1\} \\
\hat{X}^{jk} &= \{x_{jk} + \theta_1 he_1 + \theta_2 he_2 : \ -1 \le \theta_i \le 1\} \\
(\underline{f}_q^{jk})_i &= \min_{x \in \hat{X}^{jk}, u \in \Omega_u} (f_q(x,u))_i \\
(\overline{f}_q^{jk})_i &= \max_{x \in \hat{X}^{jk}, u \in \Omega_u} (f_q(x,u))_i \\
(\underline{l}_q^{jk})_i &= \min_{x \in \hat{X}^{jk}, u \in \Omega_u} (l_q(x,u))_i \\
V_q^{jk} &= V_q(x_{jk}) \\
\Delta_i V_q^{jk} &= (V_q(x_{jk} + he_i) - V_q(x_{jk}))/h \\
\Delta_{-i} V_q^{jk} &= (V_q(x_{jk}) - V_q(x_{jk} - he_i))/h
\end{aligned}
$$

where e_1 and e_2 are unit vectors along the coordinate axes, and h is the grid size.

Introduce new vector variables, $\lambda_q^{jk} \in \mathbf{R}^n$ for (j,k,q) such that $x_{jk} \in$

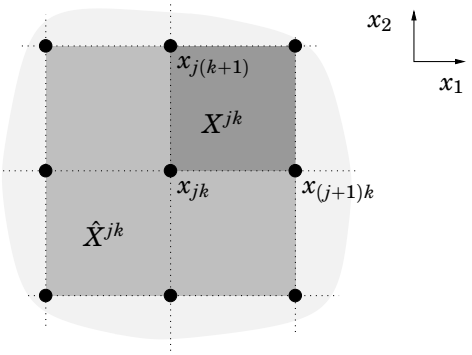

Figure 1. Illustration of X^{jk} and $\hat{X}^{jk}$.

X, $q \in Q$. The inequalities (3)-(5) can then be replaced by

$$0 \leq (\lambda_q^{jk})_1 + (\lambda_q^{jk})_2 + \underline{l}_q^{jk} \tag{6}$$

$$(\lambda_q^{jk})_{|i|} \leq (\underline{f}_q^{jk})_{|i|} \Delta_i V_q^{jk} \qquad i = -2, -1, 1, 2 \tag{7}$$

$$(\lambda_q^{jk})_{|i|} \leq (\overline{f}_q^{jk})_{|i|} \Delta_i V_q^{jk} \qquad i = -2, -1, 1, 2 \tag{8}$$

$$0 \leq V_r^{jk} - V_q^{jk} + s(x_{jk}, q, r) \qquad \forall x_{jk} \in S_{q,r} \tag{9}$$

$$0 = V_{q_f}^{00} \tag{10}$$

where (6)-(8) form a combination of backward and forward difference approximations of (3).

For $x = x_{jk} + \theta_1 h e_1 + \theta_2 h e_2 \in X^{jk}$, define the interpolating function

$$\begin{aligned} V_q(x) = {} & (1-\theta_1)(1-\theta_2) V_q^{jk} + \theta_1 (1-\theta_2) V_q^{(j+1)k} \\ & + (1-\theta_1)\theta_2 V_q^{j(k+1)} + \theta_1 \theta_2 V_q^{(j+1)(k+1)} \end{aligned} \tag{11}$$

The following result applies.

THEOREM 1—DISCRETIZATION IN $\mathbf{R}^2$
If V_q^{jk} satisfy (6)-(10) for all $q \in Q$ and for all grid points $x_{jk} \in X \subset \mathbf{R}^2$ such that X^{jk} intersects X, then the interpolating function V_q defined

by (11) satisfies (3)-(5) and, for every (x_0, q_0), $V_{q_0}(x_0)$ is a lower bound of $J(x_0, q_0)$. □

Remark 1. Any function that meet the constraints, even the trivial choice $V_q(x) = 0$, is a lower bound on the true cost. Thus, to yield useful bounds, $V_q(x)$ need to be maximized subject to (6)-(10). The maximization could be carried out in either one point, (x_0, q_0), or several points, $(x, q) \in X \times Q$, simultaneously.

For the original, non-discretized problem, the result of a maximization of $V_q(x)$ is always identical to the optimal cost, regardless if the maximization is done at a particular initial state, or by summing the values at several initial states.

However, for the discretized problem, different choices of maximization criteria may lead to different results. Fortunately, experience from examples shows that the difference between the results of a single-point and a multi-point maximization is often small, making it possible to compute the value function in a large subset of $X \times Q$ solving *one* LP.

Remark 2. The restriction $x(t) \in X$ in the optimal control problem is essential. It may happen that for some initial states x_0 there exist no admissible solutions inside X. Then the maximization of $V_{q_0}(x_0)$ can lead to arbitrarily large values.

Remark 3. The theorem is easily extended to $\mathbf{R}^n$. Define $\mathbf{j} = (j_1, j_2, \ldots, j_n)$ and exchange jk for the new multi-index $\mathbf{j}$ in the above inequalities. The limits of all summations and enumerations should also be adjusted.

Proof. Assume that $x \in X^{jk}$. Noting that $\Delta_1 V_q^{jk} = \Delta_{-1} V_q^{(j+1)k}$, $\Delta_2 V_q^{jk} = \Delta_{-2} V_q^{j(k+1)}$, the inequalities (6)-(8) taken at grid points jk, $j(k+1)$, $(j+1)k$, and $(j+1)(k+1)$ give

$$0 \le f_{q1}(x,u)\Delta_1 V_q^{jk} + f_{q2}(x,u)\Delta_2 V_q^{jk} + l_q(x,u) \tag{12}$$

$$0 \le f_{q1}(x,u)\Delta_1 V_q^{j(k+1)} + f_{q2}(x,u)\Delta_2 V_q^{jk} + l_q(x,u) \tag{13}$$

$$0 \le f_{q1}(x,u)\Delta_1 V_q^{jk} + f_{q2}(x,u)\Delta_2 V_q^{(j+1)k} + l_q(x,u) \tag{14}$$

$$0 \le f_{q1}(x,u)\Delta_1 V_q^{j(k+1)} + f_{q2}(x,u)\Delta_2 V_q^{(j+1)k} + \\ + l_q(x,u) \tag{15}$$

The gradient of V_q is given by

$$\frac{\partial V_q}{\partial x} = \begin{bmatrix} (1-\theta_2)\Delta_1 V_q^{jk} + \theta_2 \Delta_1 V_q^{j(k+1)} \\ (1-\theta_1)\Delta_2 V_q^{jk} + \theta_1 \Delta_2 V_q^{(j+1)k} \end{bmatrix}^T$$

and thus, adding (12)-(15) weighted with $(1-\theta_1)(1-\theta_2)$, $(1-\theta_1)\theta_2$, $\theta_1(1-\theta_2)$, and $\theta_1\theta_2$ respectively proves that (3) is met for x. The inequality (4) is met since V_q is a convex combination of grid points that all meet (9), and (5) is the same condition as (10). □

Note a special case in which the computational load of the local optimizations in Theorem 1 is lightened: if $f_q(x,u) = h_q(x) + g_q(x)u$ and $l_q(x,u) = o_q(x) + m_q(x)u$ while $\Omega_u = [-1,1]$, then u can be entirely eliminated from (6)-(8) by replacing $\underline{f}_q^{jk}$, $\overline{f}_q^{jk}$, and $\underline{l}_q^{jk}$ with $\underline{h}_q^{jk} \pm \underline{g}_q^{jk}$, $\overline{h}_q^{jk} \pm \overline{g}_q^{jk}$, and $\underline{o}_q^{jk} \pm \underline{m}_q^{jk}$ respectively. This will double the set of equations (6)-(8), but the functions h_q, g_q, o_q, and m_q are optimized over $\hat{X}^{jk}$ solely.

5. Computing the Control Law

Provided that the lower bound, V_q, is a good enough approximation of the optimal cost, the optimal feedback control law can be calculated as

$$\begin{cases} \hat{u}(x,q) = \underset{u\in\Omega_u}{\operatorname{argmin}} \left\{ \dfrac{\partial V_q}{\partial x} f_q(x,u) + l_q(x,u) \right\} \\ \hat{\mu}(x,q) = \underset{\mu\in\Omega_\mu \,|\, x\in S_{q,\nu}}{\operatorname{argmin}} \{V_\nu(x) + s(x,q,\nu)\} \end{cases} \tag{16}$$

where $\nu = \nu(x,q,\mu)$. Thus, the continuous input, $\hat{u}$, is computed in a standard way. The discrete input, $\hat{\mu}$, is chosen such that switching occur whenever there exist a discrete mode for which the value function has a lower value than the cost of the value function for the current mode minus the cost for switching there.

Consider the true optimal value function, $V_q^\star$. For those (x,q,r) where the optimal trajectory requires mode switching, the inequality (3) will turn to equality i.e. $V_q^\star = V_r^\star + s(x,q,r)$ (this will be shown

in Ex. 1). A consequence of this is that for (16) to describe correct switching between the modes, $s(x,q,q)$ has to be defined as $s(x,q,q) = \varepsilon > 0$ (rather than the real cost $s(x,q,q) = 0$). For $V_q^\star$, the proper control law is achieved as ε approaches 0^+. A small value of ε suffices, however, for numerical computations.

Integration of (2) along a simulated trajectory based on (16) will provide an upper bound on the optimal cost. The better the control law, the better the estimate.

6. Examples

EXAMPLE 1—A CAR WITH TWO GEARS
Consider the system

$$\begin{cases} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= g_q(x_2)u, \quad q = 1,2 \quad |u| \leq 1 \end{cases} \tag{17}$$

where $g_q(x)$ is plotted in Fig. 2. This could be seen as a crude model of a car, u being the throttle, $g_q(x)$ the efficiency for gear number q.

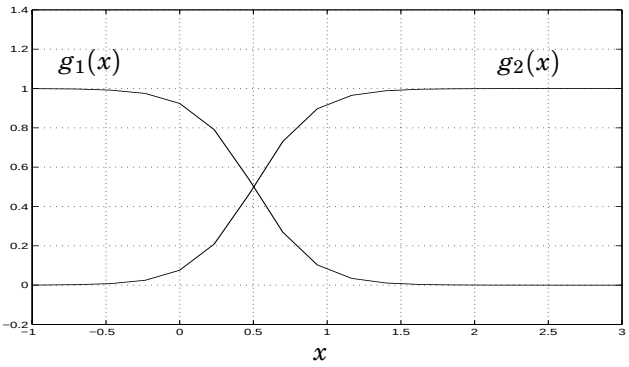

Figure 2. Gear efficiency at various speeds.

The problem is to bring (17) from $x_i = (-5,0)$, $q_i = 1$ to $x_f = (0,0)$, $q_f = 1$ in minimum time. Torque losses when using the clutch calls for

an additional penalty for gear changes. Thus, the components of (2) have been chosen as $l_1(x,u) = l_2(x,u) = 1$, $s(x,1,2) = s(x,2,1) = 0.5$.

The problem is plugged into the machinery of Section 4 and $V_q(x)$ is maximized over a region $-5.5 \leq x_1 \leq 1.0$, $-0.5 \leq x_2 \leq 3.0$.

The result is shown in Figure 3 and 4 where x_i and x_f also have been marked. The functions look rather similar, since the cost for changing gears is only 0.5. One can see that V_1 has a threshold along the line $x_2 = 1$. Figure 2 reveals that the first gear is almost useless for high speeds, leading to $V_1 = V_2 + 0.5$ for $x_2 > 1$. This is the cost for using the second gear optimally after a gear switch.

Studying Fig. 5, where $V_1 - V_2$ is plotted, the strategy for changing gears is even more obvious: there is only one discrete mode allowed under optimal control when the difference hits its maximum distance. In conformity with previous reasoning, $V_1 - V_2 = 0.5$ for $x_2 > 1$, indicating the need for a change of gears when using the first gear at high speed. Analogously, the second gear should be avoided, starting with zero speed.

A simulation of the controlled system is shown in Fig. 6, where the initial point is marked with a square. The state trajectory coincides with the one of a professional rally-driver with lousy brakes. In the beginning, maximum throttle is used on the first gear (solid line). When the speed roughly reaches the point of equal efficiency between the gears ($x_2 = 0.5$), they are switched in favor of the second gear (dashed line). At half the distance, the gas pedal is lightened to use the braking force of the engine. In the end, the first gear is used again before the origin is hit. As seen in the figure, the granularity of the discretization grid ($h = 0.18$) prevents the solution from hitting the exact origin.

□

EXAMPLE 2—ALTERNATE HEATING OF TWO FURNACES

Since the industrial power fee is determined by the highest peak of the season [Ericsson, 1997], it is desirable to spread the power consumption evenly over time. This is handled by load control, which means that the available electrical power is altered between different loads of the mill.

In this example, the temperature of two furnaces should be con-

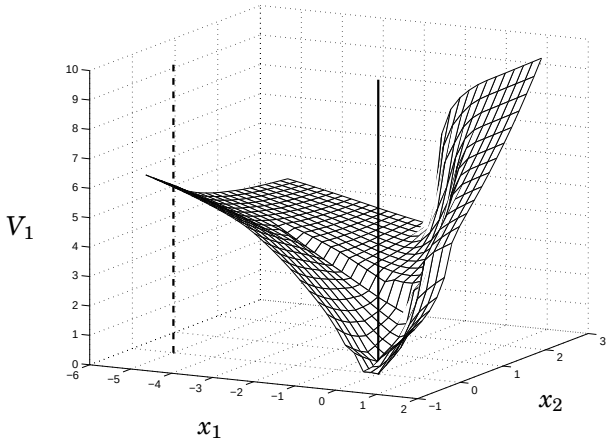

Figure 3. Plot of V_1. The initial point, x_i, is marked with a vertical dashed line, the final point, x_f, with a solid line.

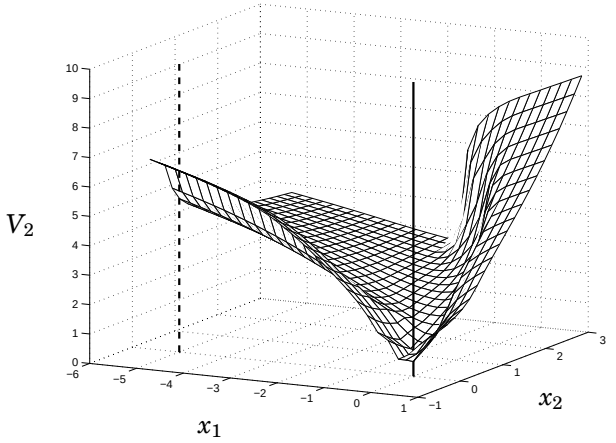

Figure 4. Plot of V_2.

trolled by alternate heating. The system has two continuous states that correspond to the temperature of the furnaces and is given by

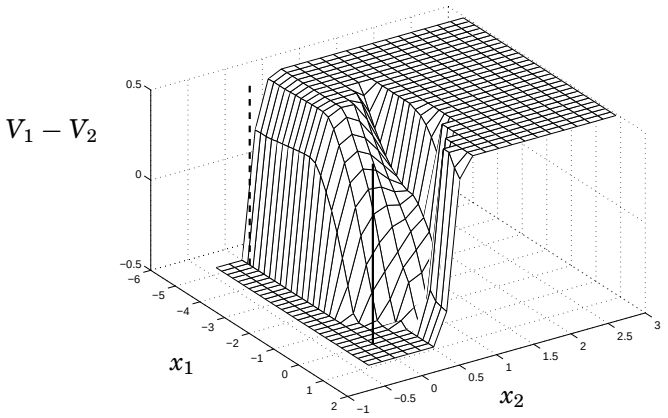

Figure 5. The difference between V_1 and V_2.

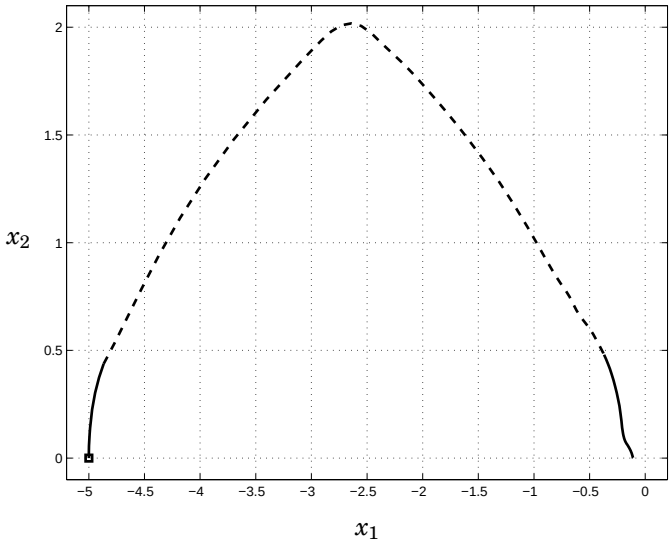

Figure 6. Phase portrait of a simulation. The solid line shows where gear number one has been used, the dashed line shows the second gear. The initial point is marked with a square.

$\dot{x} = f_q(x)$, where

$$f_1(x) = \begin{bmatrix} -x_1 + u_0 \\ -2x_2 \end{bmatrix} \quad f_2(x) = \begin{bmatrix} -x_1 \\ -2x_2 + u_0 \end{bmatrix}$$
$$f_3(x) = \begin{bmatrix} -x_1 \\ -2x_2 \end{bmatrix}$$

Thus, there are three discrete modes: $q = 1$ means that the first furnace is heated, $q = 2$ means that the second furnace is heated, $q = 3$ corresponds to no heating. The cost function to be minimized is

$$J(x_0, q_0) = \int_{t_0}^{\infty} \sum_{i=1}^{2} (x_i - c_i)^2 e^{-t} dt + \sum_{k=1}^{M} b e^{-t_k}$$

where the desired stationary temperature values are $c_1 = 1/4$, $c_2 = 1/8$ and the cost for switching the power is $b = 1/1000$. Since the furnaces can only be fed by a fixed amount of energy, u_0, it is impossible to keep them stationary at the desired temperature. Hence, the time weighting, e^{-t}, is necessary to get a bounded cost function.

If $V_q(x,t)$ is defined as the cost for starting in (x,q) at time t, then the continuous part of the general time dependent Bellman inequality can be written

$$\frac{\partial V_q(x,t)}{\partial t} + \frac{\partial V_q(x,t)}{\partial x} f_q(x,u,t) + l_q(x,u,t) \geq 0 \tag{18}$$

Rewriting the functions like $V_q(x,t) = e^{-t}\tilde{V}_q(x)$ and $l_q(x,u,t) = e^{-t}\tilde{l}_q(x,u)$ for the furnace example,
(18) becomes

$$-\tilde{V}_q(x) + \frac{\partial \tilde{V}_q(x)}{\partial x} f_q(x,u) + \tilde{l}_q(x,u) \geq 0 \tag{19}$$

Thus, the time dependence introduced in Bellman's inequality cancels and techniques similar to those presented above apply.

The optimal control results in a limit cycle as seen in Figure 7. The figure, that contains the phase portrait of the continuous states, shows

how the temperature of one furnace always decreases as the other one is heated. By alternate heating, the temperatures first climb up to, and above the set-point and then both furnaces are turned off and the state drifts towards the origin. This procedure is then repeated over and over again, making the trajectory enclose the desired steady state (marked with a circle in the figure). The trajectory has been dashed for $t \in [0, 2.8]$ to make the limit cycle clear.

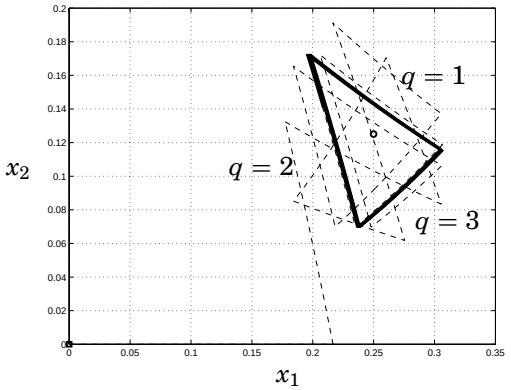

Figure 7. Phase portrait of the continuous states under optimal control when $u_0 = 0.8$. The mode number, q, has been marked for the limit cycle

Figure 8 shows what happens when the power supply is insufficient for driving both furnaces. Mode 3 is not entered since the temperature set-points are never reached.

□

7. Summary

An extended version of Bellman's inequality was discretized in this paper to compute a lower bound on the optimal cost function, using linear programming. Based on these computations, an approximation of the optimal control feedback law was derived.

Hybrid systems combine discrete and continuous dynamics. The analysis should therefore contain techniques that are well suited for computer science as well as control theory. The emphasis in this paper

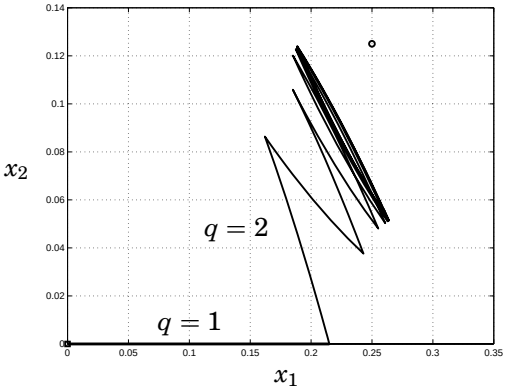

Figure 8. Phase portrait of the continuous states under optimal control when $u_0 = 0.4$.

is on the continuous part, the discrete part consisting of a few system modes. At the other end of the hybrid spectrum, where purely discrete systems are found, X will reduce to a single point. The first inequality of proposition 1 will then be superfluous. The set of inequalities given by (4), possibly large depending on Q, should be met for $S_{q,r} = \{x_f\}$. The resulting LP formulation solves the shortest-paths problem on a non-negatively weighted, directed graph — a problem that is usually attacked using Dijkstra's algorithm.

A set of MATLAB commands has been compiled by the authors to make it easy to test the above methods and implement the examples. The LP solver that is used is "PCx", developed by the Optimization Technology Center, Illinois. The MATLAB commands and a manual of usage are available free of charge upon request from the authors.

8. References

Antsaklis, P. J. and A. Nerode (1998): "Hybrid control systems: An introductory discussion to the special issue." *IEEE Transactions on Automatic Control*, **43:4**, pp. 457–460. Special issue on hybrid systems.

Bertsekas, D. P. and J. N. Tsitsiklis (1996): *Neuro-dynamic Programming*. Athena Scientific.

Branicky, M. (1998): "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems." *IEEE Transactions on Automatic Control*, **43:4**, pp. 475–482. Special issue on hybrid systems.

Branicky, M. S. and S. K. Mitter (1995): "Algorithms for optimal hybrid control." In *Proceedings of the 34th Conference on Decision & Control*. New Orleans.

Ericsson, L. (1997): *Dynamic Load Control, Power Peak Shaving Applied to a Foundry*. Lic Tech thesis, Dept. of Industrial Electrical Engineering and Automation, Lund Institute of Technology, Box 118, S-221 00 Lund, SWEDEN.

Ezzine, J. and A. H. Haddad (1989): "Controllability and observability of hybrid systems." *Int. J. Contr.*, **49**, June, pp. 2045–2055.

Grossman, R., A. Nerode, A. Ravn, and H. Rischel (1993): "Models for hybrid systems: Automata, topologies, controllability, observability." In *Hybrid Systems*, pp. 317–356. Springer.

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

Rantzer, A. (1999): "Dynamic programming via convex optimization." In *Proceedings of the IFAC World Congress*. Beijing.

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

Utkin, V. I. (1977): "Variable structure systems with sliding modes." *IEEE Transactions on Automatic Control*, **AC-22**, pp. 212–222.

Vinter, R. B. and R. M. Lewis (1978): "A necessary and sufficient condition for optimality of dynamic programming type, making no a priori assumptions on the controls." *SIAM Journal on Control and Optimization*, **16:4**, pp. 571–583.

Manual II

A Matlab Tool for Optimal Control of Hybrid Systems

Sven Hedlund and Anders Rantzer

1. Introduction

This report presents a set of MATLAB functions for the analysis and synthesis of a class of hybrid systems. The report is organized as follows: Section 2 defines the problems that this tool is designed for. Section 3 gives an overview of the available commands and the main ideas behind the computations. Few details are presented in this section, the purpose is to give the user enough understanding to be able to utilize the full functionality of the tools. Readers interested in the theory behind the computation are referred to [Hedlund and Rantzer, 1999]. Section 4 gives a complete description of the MATLAB commands of this tool, while Section 5 demonstrates the usage in some examples.

1 Disclaimer

This software and the accompanying files are distributed "as is" and without any warranties expressed or implied. Bug reports and suggestions about improvements sent to `sven@control.lth.se` are appreciated.

2. Problem Formulation

Define a hybrid system as

$$\begin{cases} \dot{x}(t) & = f_{q(t)}(x(t), u(t)) \\ q(t) & = \nu(x(t^-), q(t^-), \mu(t^-)) \end{cases} \tag{1}$$

where $x(t) \in X \subset \mathbf{R}^n$ is the state vector, $u(t) \in \Omega_u \subset \mathbf{R}^m$ is a continuous input signal of the system. There is also a discrete input, $\mu(t) \in \Omega_\mu$, which allows for the selection between N different system modes, $q(t) \in Q = \{1, 2, \ldots, N\}$. $S_{q,r}$ is a set (parameterized by q and r) such that switching from mode q to r is possible when $x \in S_{q,r} \subseteq X$. The time argument, t, will often be omitted in the sequel for readability.

The optimal control problem is to minimize the cost functional

$$J(x_0, q_0, u(\cdot), \mu(\cdot)) = \int_{t_0}^{t_f} l_q(x, u)dt + \sum_{k=1}^{M} s(x(t_k^-), q(t_k^-), q(t_k^+)) \tag{2}$$

subject to (1) while bringing the system from an initial state (x_0, q_0) at time t_0, to a final state (x_f, q_f) at time t_f, where the end time, t_f, is free. Here, M is an arbitrary number of switches occurring at times $t_0 < t_1 < t_2 < \ldots < t_M < t_f$ and $s(x, q, r)$ is an associated cost for switching from discrete state q to r, the continuous part being x just before the switch.

Sec. 4 will show that this MATLAB tool also handles exponential time weighting of the cost function.

3. Understanding the Tools

The commands available for solving the control problem are listed in Table 1. There are three main groups of programs: a group of four commands that in various ways approximate the value function of a hybrid optimal control problem, one command for deriving a control signal from the value function, and four commands for simulating hybrid systems. The last two programs listed in the table are used by the other programs.

Table 1. Commands for optimal control of hybrid systems.

command	description
ohlows	Compute a lower bound on the value function, single-point maximization
ohlowes	Compute a lower bound on the value function of an exponential time weighting problem, single-point maximization
ohlowm	Compute a lower bound on the value function, multi-point maximization
ohlowem	Compute a lower bound on the value function of an exponential time weighting problem, multi-point maximization
ohctrl	Compute a control signal, based on an approximation of the value function
ohsim	Simulate controlled system
ohsimf	Simulate controlled system, fixed time step
ohsime	Simulate controlled system with exponential time cost function
ohsimef	Simulate controlled system with exponential time cost function, fixed time step
crop	Crops a multi dimensional array
linprog	Specifies what LP solver to use in the lower bound computation programs
ohsf	"oh simulation file", used by ohsim
ohsfe	"oh simulation file", used by ohsime

1 Approximations of the Value Function

Define the value function, $V_q^\star(x)$ as

$$V_{q_0}^\star(x_0) = \min_{u \in \Omega_u, \mu \in \Omega_\mu} J(x_0, q_0, u, \mu) \tag{3}$$

Then, any set of functions $V_q : X \mapsto R$, $q = 1, 2, \ldots, N$ that satisfy

$$0 \leq \frac{\partial V_q(x)}{\partial x} f_q(x,u) + l_q(x,u) \qquad \forall x \in X,\, u \in \Omega_u,\, q \in Q \tag{4}$$

$$0 \leq V_r(x) - V_q(x) + s(x,q,r) \qquad \forall x \in S_{q,r} \;\; q, r \in Q : q \neq r \tag{5}$$

$$0 = V_{q_f}(x_f) \tag{6}$$

(with the specifications in (1) and (2)) is a lower bound on $V_q^\star(x)$[1] [Hedlund and Rantzer, 1999]. This is a hybrid version of the well known Bellman inequality.

Since the inequalities (4)-(6) are linear constraints on $V_q(x)$, maximization of $V_{q_0}(x_0)$ subject to the inequalities is an LP problem.

2 Discretization

Using a computer to find a value function that satisfies (4)-(6) for a specific control problem, a straightforward approach is to grid the state space to require the inequalities to be met at set of evenly distributed points in X. Let $e_1, e_2, \ldots, e_n$ denote the unit vectors along the coordinate axes and define the discretization vector $h \in \mathbf{R}^n$ such that h_i (the i:th component of h) is the distance between the grid points in the direction of e_i. A small part of a discretization in $\mathbf{R}^2$ around a grid point x_p is shown in Fig. 1.

Each of the value function commands applies a discretization grid like this to X. The commands handle sets, X, that are hyperrectangles in $\mathbf{R}^n$, and the user specifies the granularity of the grid by the input vector $N \in \mathbf{Z}^n$ such that the k:th component of N is the number of grid points in the direction of e_k.

An arbitrary discretization of (4)-(6) will not render a cost function that is guaranteed to be a lower bound on the optimal cost if the nature of f_q and V_q *between* the grid points is not taken into consideration.

[1] Note that the value function, $V_{q_0}^\star(x_0)$, is the cost for driving the system optimally to the final point when *starting in* mode q_0, not necessarily *staying in* this mode. This is the cost when switching is allowed.

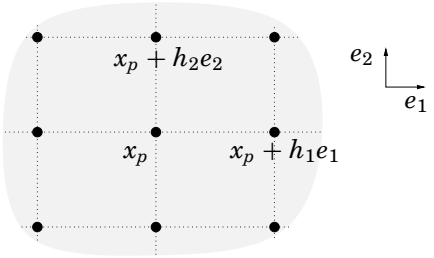

Figure 1. Illustration of the discretization grid in $\mathbf{R}^2$.

The value function commands can be set to use a method (that is presented in [Hedlund and Rantzer, 1999]) for preserving the lower bound property . For each grid point, x_p, this method requires the extremal values of f_q and l_q in a neighborhood of x_p as follows:

Define the hyperrectangle $\hat{X}^p$ surrounding grid point x_p as

$$\hat{X}^p = \{x_p + \sum_{i=1}^{n} \theta_i h_i e_i : -1 \leq \theta_i \leq 1\}. \tag{7}$$

An illustration of this set in a two dimensional space is shown in Fig. 2.

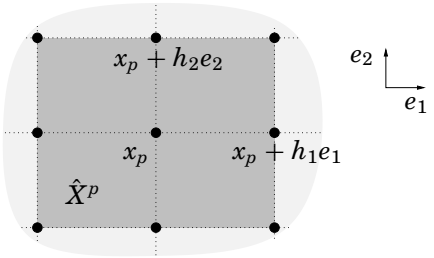

Figure 2. Illustration of $\hat{X}^p$ in $\mathbf{R}^2$.

For each grid point, x_p, the value function commands then need

$$\underline{f}_q^p(u) = \min_{x \in \hat{X}^p} f_q(x,u) \tag{8}$$

$$\overline{f}_q^p(u) = \max_{x \in \hat{X}^p} f_q(x,u) \tag{9}$$

$$\underline{l}_q^p(u) = \min_{x \in \hat{X}^p} l_q(x,u) \tag{10}$$

to form the discretized inequalities. (The extrema should be computed component wise in the vectors.)

Note that the MATLAB functions presented above can be called without requiring the true bound property, often rendering a plausible value function without forcing the user on a tricky hunt of local extrema. In addition to being more difficult to specify, the discrete inequalities that render a true lower bound sometimes can be conservative, leading to a value function that is far from the optimal one.

The extremal values of (8)-(10) still depend on a continuous parameter, namely u. The continuous control signal has to be discretized by the user in a tradeoff between accuracy and computational speed. The reason for leaving this burden to the user instead of automatically gridding the problem in u (as was done for x), is that reflections about the structure of the problem might lead to clever gridding and a reduction of the computational load.

Consider e.g. the analysis of some system with $\Omega_u = [-1,1]$. A standard gridding might have led to the approximation $\Omega_u = \{-1, -0.8, -0.6, \ldots, 1\}$. Having realized that it is a minimum time problem that will result in bang-bang control, however, the obvious choice is $\Omega_u = \{-1, 1\}$.

3 Single-point vs. Multi-point Maximization

Instead of computing the value function in one single point, (x_0, q_0), it is desirable to get an estimate for the value function in a larger subset of X in one go. This is what `ohlowm` and `ohlowem` try to do, by maximizing $V_q(x)$ in several grid points simultaneously. The region that contains the grid points for which V_q is maximized, denoted O, will of course have to obey $O \subseteq X$.

Fig. 3 illustrates the two maximization alternatives in $\mathbf{R}^2$. The optimal state trajectories have also been plotted, which raises another issue: the choice of X. The aim of this MATLAB tool is to minimize the value

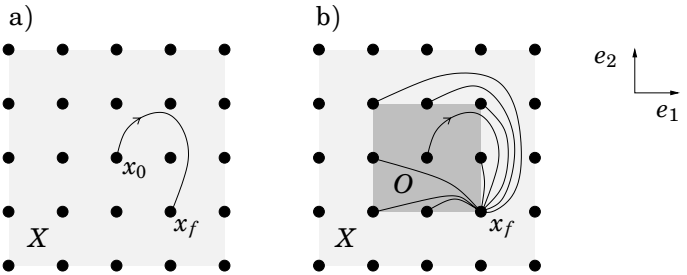

Figure 3. Various choices of maximization in $\mathbf{R}^2$ with corresponding optimal state trajectories. a) Single point maximization. b) Region maximization.

function (2) subject to the dynamics in (1), where one of the constraints is on the continuous state: $x(t) \in X$. Many control problems do not experience state constraints that are of significant importance, leading to $X = \mathbf{R}^n$. Since the discretization gives an LP consisting of a number of inequality constraints for each grid point, however, it is desirable to keep X as small as possible. The computationally best option when doing a single-point maximization would be to make X only just big enough to contain the optimal trajectory — an option that is difficult in practice since the trajectory is not known in advance. If X is chosen too small, a problem that differs from the original one is solved, leading to a higher cost. Moreover, there might not even exist a trajectory from x_0 to x_f in X, i.e. there is no feasible solution. The problem of estimating the value function in several points simultaneously requires *several* state trajectories to stay in X, which may make the choice of X even more difficult.

For the original, partly continuous, problem, the result of a maximization in two different points, (x_a, q_a) and (x_b, q_b) would be the same, regardless of whether they were maximized simultaneously or one by one. The value function of the discretized problem, however, is coupled between the grid points: if the discrete version of (4)-(5) holds in every grid point in X, then there is in general a tradeoff between the value

of $V_{q_a}(x_a)$ and $V_{q_b}(x_b)$. Thus, `ohlowm` and `ohlowem`, that maximize the sum of $V_q(x)$ in the grid points of a user specified "optimization region", $O \subseteq X$, in general give values that are lower than those that would result from maximizing each point separately. Experience from examples tells, however, that the difference is rather small, leaving the above disadvantage beaten by the benefit of receiving the value function in a large region solving *one* LP.

4 Exponential Time Weighting

The methods presented above, also can be used for problems with exponential time weighting of the cost function. Define the cost function $J_e(x_0, q_0, u(\cdot), \mu(\cdot))$ as

$$J_e(x_0, q_0, u(\cdot), \mu(\cdot)) = \int_{t_0}^{\infty} \tilde{l}_q(x,u)e^{-at}dt + \sum_{k=1}^{M} \tilde{s}(x(t_k^-), q(t_k^-), q(t_k^+))e^{-at_k} \tag{11}$$

where $a \in \mathbf{R}^+$. (The other parameters are defined analogously to (2).) If $V_q(x,t)$ is defined as the cost for starting in (x,q) at time t, then the continuous part of the general time dependent Bellman inequality can be written

$$\frac{\partial V_q(x,t)}{\partial t} + \frac{\partial V_q(x,t)}{\partial x} f_q(x,u,t) + l_q(x,u,t) \geq 0 \tag{12}$$

Rewriting the functions like $V_q(x,t) = e^{-at}\tilde{V}_q(x)$ and $l_q(x,u,t) = e^{-at}\tilde{l}_q(x,u)$, (12) becomes

$$-a\tilde{V}_q(x) + \frac{\partial \tilde{V}_q(x)}{\partial x} f_q(x,u) + \tilde{l}_q(x,u) \geq 0 \tag{13}$$

Thus, the time dependence introduced in the Bellman inequality cancels and techniques similar to those presented above apply. The function `ohlowes` and `ohlowem` implement a discretized version of (13). The former function perform a single-point maximization, the latter one a multi-point maximization.

5 Computing the Feedback Control Law

Provided that the lower bound, V_q, is a good enough approximation of the optimal cost, the optimal feedback control law can be calculated as

$$\begin{cases} \hat{u}(x,q) & = \underset{u\in\Omega_u}{\operatorname{argmin}} \left\{ \dfrac{\partial V_q}{\partial x} f_q(x,u) + l_q(x,u) \right\} \\ \hat{\mu}(x,q) & = \underset{\mu\in\Omega_\mu \,|\, x\in S_{q,\nu}}{\operatorname{argmin}} \{V_\nu(x) + s(x,q,\nu)\} \end{cases} \tag{14}$$

where $\nu = \nu(x,q,\mu)$. Thus, the continuous input, $\hat{u}$, is computed in a standard way. The discrete input, $\hat{\mu}$, is chosen such that switching occur whenever there exist a discrete mode for which the value function has a lower value than the cost of the value function for the current mode minus the cost for switching there. The function `ohctrl`, that uses (14) to compute the control signal in a mesh of points, will in most practical cases take the result from `ohlowm` or `ohlowem` as input.

6 Simulation

The simulation commands take a hybrid system with a cost function and the associated control law as input and return the resulting trajectories, $x(t)$, $q(t)$, $u(t)$, and $J(t)$ or $J_e(t)$. The basic functions for simulations are `ohsim` and `ohsime`, but there also exist faster, less accurate fixed time step size versions, `ohsimf` and `ohsimef`.

4. Command Reference

This section describes the commands in detail. Being very similar to each other, some of the commands of Table 1 are grouped into the same entry on the following pages. The commands `ohsf` and `ohsfe` are not found in this section, since they are of little interest to the standard user.

Note that all of the input parameters that are vectors, should be entered as column vectors.

crop

Purpose

To crop a multi dimensional array.

Synopsis

```
[Vc, xvc] = crop(V, xv, NewX)
```

Description

Having computed a value function, `V`, using `ohlowm` or `ohlowem`, the function `crop` is useful for extracting the relevant data of `V` as is explained below. The input parameter `V` and `xv`, that are outputs from the value function computing commands, contain the following:

`xv` The input parameter `xv` is a struct of vectors that gives information about the discretization. `xv{k}` is a vector with `N(k)` equidistant points in the e_k direction such that `xv{k}(1)=xmin(k)` and `xv{k}(N(k))=xmax(k)`.

`V` is an $(n+1)$-dimensional matrix that corresponds to the value function as follows. Let a grid point x_p be defined by its index-vector `p=[p1, p2, ..., pn]'` (all the indices are positive integers) such that
$x_p =$`[xv{1}(p1), xv{2}(p2), ..., xv{n}(pn)]'`. Then,
$V_q(x_p) =$`V(p1, p2, ..., pn, q)`

Thus, a lower bound on the value function, $V_q(x)$, is returned for all $x \in X$. Since the maximization of $V_q(x)$ is performed over a smaller region $O \subseteq X$ (cf. Section 3), the value function that `ohlowm` and `ohlowem` return is of little use for $x \in X \backslash O$. The input parameter `NewX` allows the user to remove $x \in X \backslash O$:

`NewX = [xmin; xmax]` `newX` is the X-region that should be kept in the cropped matrix. `xmin` is a vector where each component is the lowest value of the corresponding state in `NewX`, `xmax` is a vector containing the highest values of x in `newX`.

The output parameters `Vc` and `xvc` are the cropped versions of `V` and `xv` respectively.

See Also

`ohlowm, ohlowem`

linprog

Purpose

Let the user specify what LP solver to use.

Synopsis

```
[x, z] = linprog(A,b,c)
```

Description

The command `linprog` is called by the value function commands (`ohlowm`, `ohlows`, `ohlowem`, and `ohlowes`) to solve linear programs. The default file `linprog.m` forwards the LP solving request to the MATLAB program `PCx`.[2] The user may want to rewrite `linprog` to call another LP solver.

Parameters

The input parameters are a matrix, `A`, and two column vectors, `b` and `c`. The output parameter `x` is the vector that minimizes $c^T x$ subject to $Ax \leq b$. The output parameter `z` is the vector that solves the dual problem of maximizing $b^T z$ subject to $A^T z = c$, $z \leq 0$.

See Also

`ohlowm, ohlows, ohlowem, ohlowes`

[2] `PCx` has been developed at the Optimization Technology Center, `http://www-fp.mcs.anl.gov/otc/Tools/PCx/index.html` (valid in August, 1999)

ohctrl

Purpose

Derive a feedback control law from a value function approximation.

Synopsis

```
[U, Q] = ohctrl(f,l,s,uv,V,xv,swtol)
```

Description

`ohctrl` derives a feedback control law from a value function approximation returned by `ohlowm` or `ohlowem`.

Parameters

`f` The string `f` contains the name of an m-file that describes a system of differential equations such that `f(x, q, u)` gives the dynamics of $f_q(x,u)$ in (1).

`l` The string `l` contains the name of an m-file on the form `l(x, q, u)` with the input parameters corresponding to the parameters of either $l_q(x,u)$ in (2) or $\tilde{l}_q(x,u)$ in (11).

`s` The string `s` contains the name of an m-file on the form `s(x, q1, q2)` with the input parameters corresponding to the parameters of either $s(x,q_1,q_2)$ in (2) or $\tilde{s}(x,q_1,q_2)$ in (11).

`uv` is a column vector that contains all possible values of u. The control signal u is continuous in the original problem, but the user has to approximate it by a discrete set (cf. 2). Use an empty vector (`uv = []`) as a place holder if there is no continuous input.

`V` is an $(n+1)$-dimensional matrix that corresponds to the value function as follows. Let a grid point x_p be defined by its index-vector `p=[p1, p2, ..., pn]'` (all the indices are positive integers) such that
x_p =`[xv{1}(p1), xv{2}(p2), ..., xv{n}(pn)]'`. Then,
$V_q(x_p)$ =`V(p1, p2, ..., pn, q)`

`xv` is a struct of vectors that gives information about the discretization. `xv{k}` is a vector with `N(k)` equidistant points in the e_k direction such that `xv{k}(1)=xmin(k)` and `xv{k}(N(k))=xmax(k)`.

`swtol` compensates for numerical inaccuracy. Discrete mode switching from q_i to q_j will be enforced if $V_j - V_i + (1-\texttt{swtol}) \cdot s(x,i,j) \leq 0$. If not specified, this parameter is given the default value 0.01.

`U` The output parameter `U` is an $(n+1)$-dimensional matrix that represents the control law for u such that $u = u(x,q)$. Defining x_p as above, the control law is $u(x_p,q)$ =`U(p1, p2, ..., pn, q)`

`Q` The output parameter `Q` is an $(n+1)$-dimensional matrix that represents the switching strategy. Defining x_p as above, `Q(p1, p2, ..., pn, q1)`$= q_2$ implies switching to mode q_2 from (x_p, q_1).

See Also

`ohlowm, ohlowem`

ohlowem & ohlowes

Purpose

Compute a lower bound on the value function of a problem with a cost function that has exponential time weighting.

Synopsis

```
[V, xv, W] = ohlowem(f, l, s, a, uv, O, XQ, N, tb)
[V, xv, W] = ohlowes(f, l, s, a, uv, xq0, XQ, N, tb)
```

Description

Both of these two commands, compute an approximation of the value function implied by (11). `ohlowes` computes the cost for bringing (1) from (x_0, q_0) to (x_f, q_f), while `ohlowem` computes the value function for a region, $\{(x,q) : x \in O, q \in Q\}$[3]. The commands can be forced to bound the value function from below.

Parameters

The parameters that are not found below are described under `ohlowm` and `ohlows`

`l` The string `l` contains the name of an m-file that corresponds to $\tilde{l}_q(x,u)$ in (11). The input parameters are the same as for `l` described under the `ohlow`-commands.

`s` The string `s` contains the name of an m-file of the switching such that `s(x, q1, q2)` corresponds to $\tilde{s}(x, q_1, q_2)$ in (11)

`a` is the exponential time weight a in (11).

`V` The structure of the output parameter `V` is the same as for `V` returned by `ohlowm` and `ohlows`. The difference is that the matrix `V` that is returned by `ohlowem` and `ohlowes` corresponds to $\tilde{V}_q$ in (13).

See Also

`ohlows, ohlowm, linprog`

[3] Cf. Section 3 for further details.

ohlowm & ohlows

Purpose

Compute a lower bound on the value function

Synopsis

```
[V, xv, W] = ohlowm(f, l, s, uv, O, xqf, XQ, N, tb)
[V, xv, W] = ohlows(f, l, s, uv, xq0, xqf, XQ, N, tb)
```

Description

Both of the two commands, in the sequel referred to as the ohlow-commands, compute an approximation of the value function implied by (2). `ohlows` computes the cost for bringing (1) from (x_0, q_0) to (x_f, q_f), while `ohlowm` computes the value function for a region, $\{(x,q) : x \in O, q \in Q\}$[4]. Both commands can be forced to bound the value function from below.

Parameters

`f` The string `f` contains the name of an m-file with the hybrid dynamics corresponding to $f_q(x,u)$ in (1). This m-file should take at least three input arguments depending on the input `tb`. If `tb`$= 0$, then `f` is a system of differential equations such that `f(x, q, u)` gives the dynamics of $f_q(x,u)$. If `tb`$\neq 0$, then `f` will have to accept five input parameters, `f(x, q, u, h, vmode)`. The two additional parameters `h` and `vmode` are needed to compute a true lower bound and should allow the ohlow-commands to request various outputs from `f`: the n-dimensional vector `h`, that the commands provides upon calling `f`, corresponds to the granularity of the discretization grid (cf. Section 2), such that `h(k)` is the distance between the grid points in the e_k-direction.

[4] Cf. Section 3 for further details.

The parameter vmode choose the output according to the following table:

vmode	Desired output
-1	$\min_{x \in \hat{X}^p} f_q(x_p, q, u)$
$+1$	$\max_{x \in \hat{X}^p} f_q(x_p, q, u)$

It is convenient to allow f and l to take a variable number of input arguments when computing true bounds. If they are programmed to return the nominal (non-extremal) values when called with only three input parameters, they can be used by ohctrl and the simulation programs as well.

l The string l contains the name of an m-file that corresponds to $l_q(x,u)$ in (2). This m-file should take at least three input arguments depending on the input tb. If tb$= 0$, the structure of l is l(x, q, u) analogously to the function f. If a true bound is requested, i.e. tb$\neq 0$, l should accept four input parameters, l(x, q, u, h). The parameter h is the same as described for the input f. Note that l does not require the input "vmode", since this function should only return $\min_{x \in \hat{X}^p} l_q(x_p, q, u)$.

s The string s contains the name of an m-file for the switching such that s(x, q1, q2) corresponds to $s(x, q_1, q_2)$ in (2)

uv is a column vector that contains all possible values of u. The control signal u is continuous in the original problem, but the user has to approximate it by a discrete set (cf. 2). Use an empty vector (uv = []) as a place holder if there is no continuous input.

xq0 = [x0; q0] for ohlows where (x_0, q_0) is the initial state.

O specifies the region of maximization, $O \subseteq X$, for the multi-point maximizing function ohlowm (cf. Section 3). O = [omin; omax], where omin is an n-dimensional vector where each component is the lowest value of the corresponding state in O and omax is a vector containing the highest values of x in O.

xqf = [xf; qf] where (x_f, q_f) is the desired final state. Since the problem is discretized into a mesh of points, the final state that is used in the algorithm will become the grid point that is closest

to (x_f, q_f). Note that choosing `XQ` and `N` such that the final state appears in an exact grid point often leads to considerably better results (see also the example of Section 5).

A set of several acceptable final states can be specified by replacing the variable `xqf` with a function `isfinal([x; q])` that for any possible state (x, q) returns a non-zero value if (x, q) is contained in the set of final states.

`XQ` = `[xmin; xmax; Q]` where `xmin` is an n-dimensional vector where each component is the lowest value of the corresponding state in X, `xmax` is a vector containing the highest values of x in X, and `Q` is the number of discrete modes.

`N` allows the user to specify the granularity of the discretization grid. `N` is an n-dimensional column vector such that `N(k)` is the number of grid points in the direction of e_k.

`tb` is used to choose whether to compute a true lower bound (`tb`$\neq 0$) or not (`tb`$= 0$)

`xv` The output parameter `xv` is a struct of vectors that gives information about the discretization. `xv{k}` is a vector with `N(k)` equidistant points in the e_k direction such that `xv{k}(1)=xmin(k)` and `xv{k}(N(k))=xmax(k)`.

`V` is an $(n+1)$-dimensional matrix that corresponds to the value function as follows. Let a grid point x_p be defined by its index-vector `p=[p1, p2, ..., pn]'` (all the indices are positive integers) such that
$x_p =$`[xv{1}(p1), xv{2}(p2), ..., xv{n}(pn)]'`. Then,
$V_q(x_p) =$`V(p1, p2, ..., pn, q)`

`W` `W` is an $(n+1)$-dimensional matrix that reflects the dual variables of the solution to the LP that is solved. `W(p1, p2, ..., pn, q)` is an aggregation of the dual variables involved in the discretization of (4) for the state (x_p, q), where x_p is defined as in the description of `V`.

See Also

`ohlowes, ohlowem, linprog`

ohsim & ohsimf

Purpose

Simulate a controlled hybrid system.

Synopsis

```
[t,x,q,u,J] = ohsim(f,l,s,U,Q,xv,xq0,tspan,xqf,rxftol)
[t,x,q,u,J] = ohsimf(f,l,s,U,Q,xv,xq0,dt,tspan,xqf,rxftol)
```

Description

`ohsim` and `ohsimf` simulate a hybrid system using the feedback control law returned by `ohctrl`. The difference between the commands is that `ohsim` calls an ODE solver in MATLAB, while `ohsimf` uses a fixed time step to allow faster (and less accurate) simulations.

Parameters

`f` The string `f` contains the name of an m-file that describes a system of differential equations such that `xdot = f(x, q, u)` gives the dynamics of $f_q(x,u)$ in (1).

`l` The string `l` contains the name of an m-file that describes a cost such that `l(x, q, u)` corresponds to $l_q(x,u)$ in (2).

`s` The string `s` contains the name of an m-file that describes a cost such that `s(x, q, u)` corresponds to $s_q(x,u)$ in (2).

`U` is an $(n+1)$-dimensional matrix returned by `ohctrl` that gives the control law for u.

`Q` is an $(n+1)$-dimensional matrix returned by `ohctrl` that gives the switching strategy.

`xv` is a struct of vectors that gives information about the discretization of the control laws contained in `U` and `Q`. This is the same parameter as the one used in `ohctrl`.

`xq0` `= [x0; q0]` for `ohlows` where (x_0, q_0) is the initial state.

`dt` is the fixed time step used by `ohsimf`.

`tspan` The parameters `tspan`, `xqf`, `rxftol` set various stopping criteria. `tspan` is a vector specifying the interval of integration `[t0; tfinal]`. No simulation time will pass `tfinal`. The simulation could, however, finish earlier if the final point, `xqf = [xf; qf]` is reached. It is considered to be reached when $q =$`qf` and $\texttt{xf} - \texttt{rxftol}h \leq x \leq \texttt{xf} + \texttt{rxftol}h$, where $h \in \mathbf{R}^n$ is a vector representing the grid size (cf. Sec. 2). If `rxftol` is omitted, the default value 0.5 is used. If `xqf` is omitted, then the time is used as the only stopping criterion. A set of several acceptable final states can be specified by replacing the variable `xqf` with a function `isfinal([x; q])` that for any possible state (x, q) returns a non-zero value if (x, q) is contained in the set of final states.

`xqf` cf. `tspan`

`rxftol` cf. `tspan`

`t` The output vectors `x`, `q`, `u`, and `J` contain the trajectories of the solution. Each entry of these vectors correspond to a time returned in the column vector `t`.

`x` is a vector that contains the trajectory of the continuous state, x. Each entry in `x` corresponds to a time in `t`.

`q` is a vector that contains the trajectory of the discrete mode, q. Each entry in `q` corresponds to a time in `t`.

`u` is a vector that contains the trajectory of the control signal u. Each entry in `u` corresponds to a time in `t`.

`J` is a vector that contains the accumulated cost along the solution trajectory. Each entry in `J` corresponds to a time in `t`.

See Also

`ohctrl`, `ohsime`, `ohsimef`

ohsime & ohsimef

Purpose

Simulate a controlled hybrid system, the control of which has been derived from a cost function with exponential time weighting.

Synopsis

```
[t,x,q,u,Je] = ohsime(f,l,s,a,U,Q,xv,xq0,tspan)
[t,x,q,u,Je] = ohsimef(f,l,s,a,U,Q,xv,xq0,dt,tspan)
```

Description

`ohsime` and `ohsimef` simulate a controlled hybrid system, for which the control law returned by `ohctrl` has been derived from a cost function on the form (11). The difference between the commands is that `ohsime` calls an ODE solver in MATLAB, while `ohsimef` uses a fixed time step to allow faster (and less accurate) simulations.

Parameters

The input parameters are described below. The output parameters are the same as the output from `ohsim` and `ohsimf`.

`f` The string `f` contains the name of an m-file that describes a system of differential equations such that `f(x, q, u)` gives the dynamics of $f_q(x,u)$ in (1).

`l` The string `l` contains the name of an m-file that describes a cost such that `l(x, q, u)` corresponds to $\tilde{l}_q(x,u)$ in (11).

`s` The string `s` contains the name of an m-file that describes a cost such that `s(x, q, u)` corresponds to $\tilde{s}_q(x,u)$ in (11).

`a` is the exponential time weight a in (11).

`U` is an $(n+1)$-dimensional matrix returned by `ohctrl` that gives the control law for u.

`Q` is an $(n+1)$-dimensional matrix returned by `ohctrl` that gives the switching strategy.

`xv` is a struct of vectors that gives information about the discretization of the control laws contained in `U` and `Q`. This is the same parameter as the one used in `ohctrl`.

`xq0` = `[x0; q0]` for `ohlows` where (x_0, q_0) is the initial state.

`dt` is the fixed time step used by `ohsimef`.

`tspan = [t0; tfinal]` is a vector specifying the interval of integration.

See Also

`ohctrl, ohsim, ohsimf`

5. Examples

In order to clarify the usage of the commands in this report, two examples are presented in this section. These examples contain the essential code, i.e. one should be able to reproduce similar results by entering the lines marked with the MATLAB prompt (>>) into MATLAB. Some of the figures are drawn using certain line types or contain lines that were added to make the discussion about certain phenomena clearer. The code for this pedagogic bonus has been omitted below.

1 A car with two gears

Consider the system

$$\begin{cases} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= g_q(x_2)u, \quad q = 1,2 \quad |u| \leq 1 \end{cases} \tag{15}$$

where $g_q(x)$ is plotted in Fig. 4. This could be seen as a crude model of a car, u being the throttle, $g_q(x)$ the efficiency for gear number q.

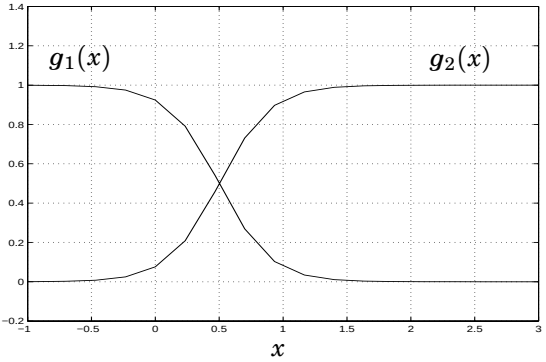

Figure 4. Gear efficiency at various speeds.

The problem is to bring (15) from $x_i = (-5, 0)$, $q_i = 1$ to $x_f = (0, 0)$, $q_f = 1$ in minimum time. Torque losses when using the clutch calls for an additional penalty for gear changes. Thus, the components of (2) have been chosen as $l_1(x, u) = l_2(x, u) = 1$, $s(x, 1, 2) = s(x, 2, 1) = 0.5$.

We start by writing the functions that define the system: f_q is entered into the file `car_f.m`, l_q into `car_l.m`, and finally s into `car_s.m`. We will not compute a true lower bound in this example. The extremal computations that would be needed for true bound purposes are included in the files anyway, to show how this could be done.

car_f.m

```
function y = car_f(x,q,u,h,vmode);
if (nargin > 3)
                        %%% perform extremal computations %%%
  nx2 = x(2)-h(2);        % min value of x2 over a square
  xx2 = x(2)+h(2);        % max value of x2 over a square
  if (q==1)
    if (vmode == -1)      % component-wise minimization
      y = [nx2; 1*sigmf(xx2, [-5, 0.5])*u];
    elseif (vmode == 1)   % component-wise maximization
      y = [xx2; 1*sigmf(nx2, [-5, 0.5])*u];
    end;
  elseif(q==2)
    if (vmode == -1)      % component-wise minimization
      y = [nx2; 1*sigmf(nx2, [5, 0.5])*u];
    elseif(vmode == 1)    % component-wise maximization
      y = [xx2; 1*sigmf(xx2, [5, 0.5])*u];
    end;
  end;
else
                        %%% use the nominal value %%%
  if (q==1)
    y = [x(2); 1*sigmf(x(2), [-5, 0.5])*u];
  elseif(q==2)
    y = [x(2); 1*sigmf(x(2), [5, 0.5])*u];
  end;
end;
```

car_l.m

```
function y = car_l(x,q,u,h);
% For this example, l is the same regardless of the input.
na = nargin;    % dummy-line to allow variable number of
                % inputs
y  = 1;
```

car_s.m

```
function y = car_s(x, q1, q2);
y = (q1~=q2)*0.5;            % The cost for switching is 0.5
```

Having entered these functions, we are ready to call `ohlowm` to get an approximation of the value function. Note that this is a minimum time problem that will lead to bang-bang control, which means that we save computational time by letting $\Omega_u = \{-1, 1\}$.

```
>> uv   = [-1; 1];

>> xf   = [0;0];
>> qf   = 1;
>> xqf  = [xf; qf];

>> xmin = [-6.5; -1.5];
>> xmax = [5; 5.5];

>> omin = [-5.5; -0.5];
>> omax = [1; 3.0];
>> O    = [omin; omax];

>> N    = [53;41];

>> odx  = [+0.0865; -0.0750]; % put the origin in a
>> xmin = xmin + odx;         % grid point
>> xmax = xmax + odx;
>> Q    = 2;
>> XQ   = [xmin; xmax; Q];
```

```
>> [V,xv] = ohlowm('car_f','car_l','car_s',uv,0,xqf,XQ,N,0);
>> [Vc,xvc] = crop(V,xv,0);
```

The three lines commented "put the origin in a grid point" are there to make the final state appear in a grid point. (E.g. having 53 equidistant grid points in the e_1 direction, ranging from $x_1 = -6.5$ to $x_1 = 5$, simple calculations show that adding 0.0865 will make one of the points appear at $x_1 = 0$.) Experience tells that placing the final state in an exact grid point often leads to considerably better results.

The value functions are plotted by typing

```
>> figure;
>> mesh(xvc{1},xvc{2},Vc(:,:,1)');
>> title('V_1');
>> xlabel('x1')
>> ylabel('x2')

>> figure;
>> mesh(xvc{1},xvc{2},Vc(:,:,2)');
>> title('V_2');
>> xlabel('x1')
>> ylabel('x2')
```

and the result is shown in Figure 5 and 6 where x_i and x_f also have been marked. The functions look rather similar, since the cost for changing gears is only 0.5. One can see that V_1 has a threshold along the line $x_2 = 1$. Figure 4 reveals that the first gear is almost useless for high speeds, leading to $V_1 = V_2 + 0.5$ for $x_2 > 1$. This is the cost for using the second gear optimally after a gear switch.

We also compute a control law and use it in simulations

```
>> [U,Q] = ohctrl('car_f','car_l','car_s',uv,Vc,xvc);

>> x0   = [-5;0];
>> q0   = 1;
>> xq0  = [x0; q0];
```

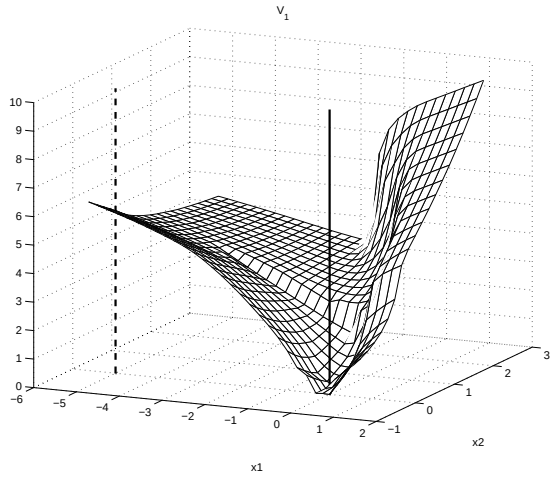

Figure 5. Plot of V_1. The initial point, x_i, is marked with a vertical dashed line, the final point, x_f, with a solid line.

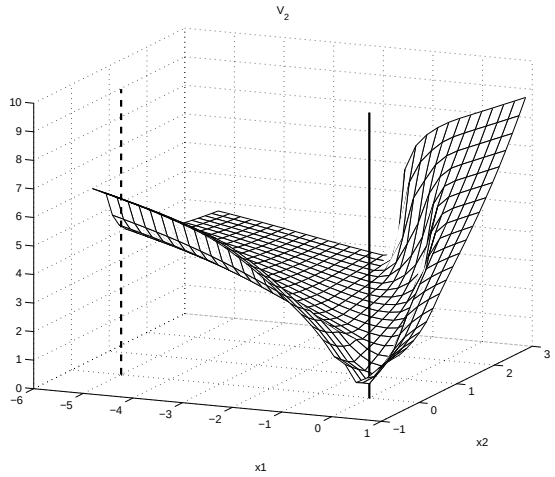

Figure 6. Plot of V_2.

```
>> tend = 8;
>> [tv,xv2,qv] = ...
   ohsim('car_f','car_l','car_s',U,Q,xvc,xq0,[0;tend],xqf);
```

The trajectory is plotted by typing

```
>> figure;
>> plot(xv2(:,1),xv2(:,2));
>> title('Phase plane');
>> xlabel('x_1');
>> ylabel('x_2');
>> grid;
```

and the result is shown in Fig. 7, where the initial point is marked with a square. The state trajectory coincides with the one of a professional rally-driver with lousy breaks. In the beginning, maximum throttle is used on the first gear (solid line). When the speed roughly reaches the point of equal efficiency between the gears ($x_2 = 0.5$), they are switched in favor of the second gear (dashed line). At half the distance, the gas pedal is lightened to use the breaking force of the engine. In the end, the first gear is used again before the origin is hit. As seen in the figure, the granularity of the discretization grid ($h_1 = 0.22$, $h_2 = 0.18$) prevents the solution from hitting the exact origin.

Information about the optimal trajectory can also be found in the dual variables. Note that the following code makes a single point maximization.

```
>> [Vs,xvs,Ws] = ...
   ohlows('car_f','car_l','car_s',uv,xq0,xqf,XQ,N,0);
>> Nxmin       = [-6; -0.5];
>> Nxmax       = [1; 3];
>> NX          = [Nxmin; Nxmax];
>> [Ws,xvs]    = crop(Ws,xvs,NX);

>> figure;
>> mesh(xvs{1},xvs{2},Ws(:,:,1)');
>> title('W_1');
```

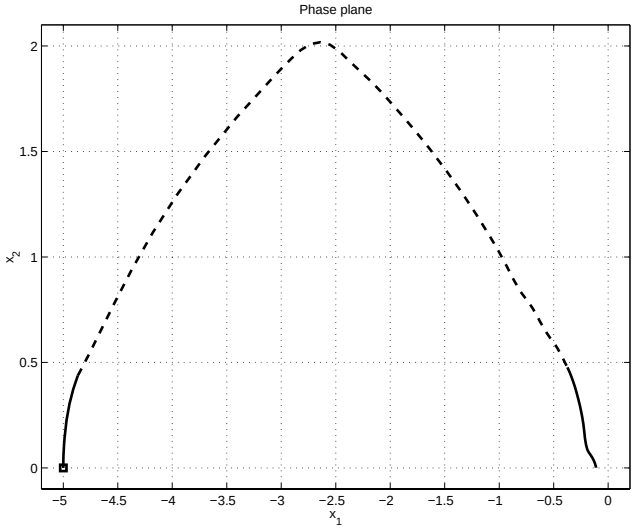

Figure 7. Phase portrait of a simulation. The solid line shows where gear number one has been used, the dashed line shows the second gear. The initial point is marked with a square.

```
>> xlabel('x1');
>> ylabel('x2');

>> figure;
>> mesh(xvs{1},xvs{2},Ws(:,:,2)');
>> title('W_2');
>> xlabel('x1');
>> ylabel('x2');
```

The optimal trajectory is easily found in Figs. 8 and 9. W_1 shows where the first gear is used, and W_2 where the second is used.

2 Alternate heating of two furnaces

Since the industrial power fee is determined by the highest peak of the season, it is desirable to spread the power consumption evenly over

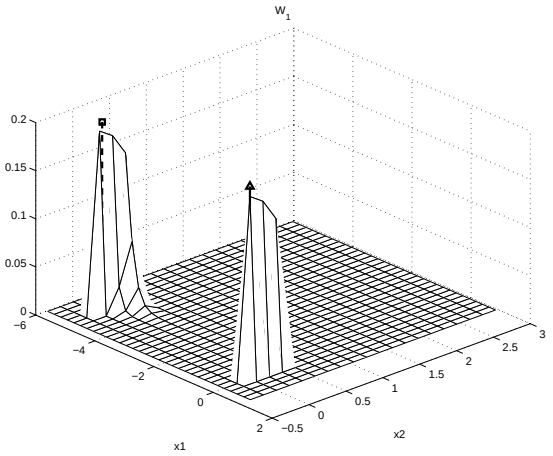

Figure 8. Plot of W_1. The initial point, x_i, is marked with a vertical dashed line, the final point, x_f, with a solid line.

time. This is handled by load control, which means that the available electrical power is altered between different loads of the mill.

In this example, the temperature of two furnaces should be controlled by alternate heating. The system has two continuous states that correspond to the temperature of the furnaces and is given by $\dot{x} = f_q(x)$, where

$$f_1(x) = \begin{bmatrix} -x_1 + u_0 \\ -2x_2 \end{bmatrix} \quad f_2(x) = \begin{bmatrix} -x_1 \\ -2x_2 + u_0 \end{bmatrix}$$

$$f_3(x) = \begin{bmatrix} -x_1 \\ -2x_2 \end{bmatrix}$$

Thus, there are three discrete modes: $q = 1$ means that the first furnace is heated, $q = 2$ means that the second furnace is heated, $q = 3$ corresponds to no heating. The cost function to be minimized is

$$J(x_0, q_0) = \int_{t_0}^{\infty} \sum_{i=1}^{2} (x_i - c_i)^2 e^{-t} dt + \sum_{k=1}^{M} b e^{-t_k}$$

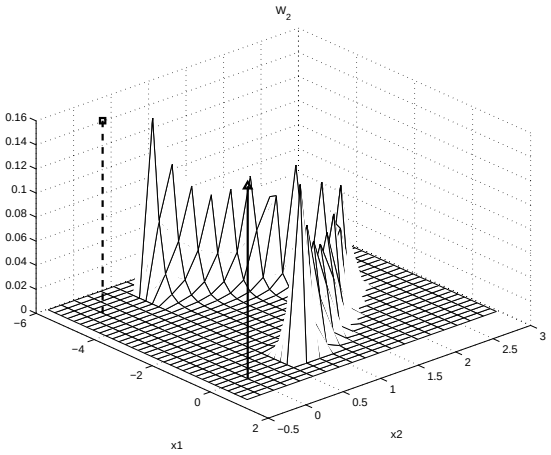

Figure 9. Plot of W_2.

where the desired stationary temperature values are $c_1 = 1/4$, $c_2 = 1/8$ and the cost for switching the power is $b = 1/1000$. Since the furnaces can only be fed by a fixed amount of energy, u_0, it is impossible to keep them stationary at the desired temperature. Hence, the time weighting, e^{-t}, is necessary to get a bounded cost function.

We start by writing the functions that define the system: f_q is entered into the file `fu_f.m`, l_q into `fu_l.m`, and finally s into `fu_s.m`.

fu_f.m

```
function y = fu_f(x,q,u)
u0 = 0.8; % 0.8 will make the system enter mode 3 sometimes,
          % 0.4 prevents it
switch (q)
case 1,                          % Heating furnace no. 1
  y = [-x(1)+u0; -2*x(2)];
case 2,                          % Heating furnace no. 2
  y = [-x(1); -2*x(2)+u0];
case 3,                          % No heating
  y = [-x(1); -2*x(2)];
```

```
end;
```

fu_l.m

```
function y = fu_l(x,q,u)
y = (x(1)-0.25)^2+(x(2)-0.125)^2;
```

fu_s.m

```
function y = fu_s(x, q1, q2);
y = (q1~=q2)*0.001;        % The cost for switching is 0.001
```

With these functions, we are ready to call `ohlowem`. Note that we have no continuous input, u, in this example.

```
>> uv    = [];
>> omin  = [-0.05; -0.05];
>> omax  = [0.40; 0.20];
>> O     = [omin; omax];
>> xf    = [0.25; 0.125];
>> qf    = 3;
>> xqf   = [xf; qf];
>> xmin  = [-0.10; -0.10];
>> xmax  = [0.50; 0.30];
>> Q     = 3;
>> XQ    = [xmin; xmax; Q];
>> N     = [21; 21];
>> [V, xv]   = ohlowem('fu_f','fu_l','fu_s',1,uv,O,XQ,N,O);
>> [Vc, xvc] = crop(V,xv,O);
```

The control law is derived and simulation is performed by calling `ohsime` this time

```
>> [Um, Qm] = ohctrl('fu_f','fu_l','fu_s',uv,Vc,xvc);

>> x0   = [0; 0];
>> q0   = 3;
>> xq0  = [x0; q0];
```

```
>> tend = 6;
>> [tv,xv2,qv] = ...
   ohsime('fu_f','fu_l','fu_s',1,Um,Qm,xvc,xq0,[0;tend]);
```

and the result is plotted in Fig. 10, which shows a time plot of the states and Fig. 11, which shows a phase portrait. The figures clearly show how the temperature of one furnace always decreases as the other one is heated. By alternate heating, the temperatures first climb up to, and above the set-point and then both furnaces are turned off and the state drifts towards the origin. This procedure is then repeated over and over again, making the trajectory enclose the desired steady state (marked with a circle in the phase portrait). The trajectory has been dashed for $t \in [0, 3.5]$ in Fig. 11 to make the limit cycle clear.

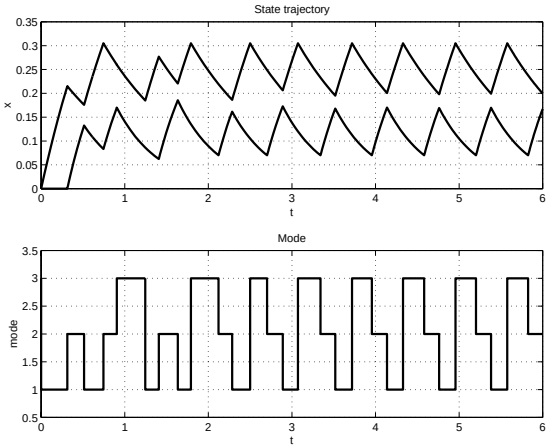

Figure 10. Time plot of the trajectories in the furnace example.

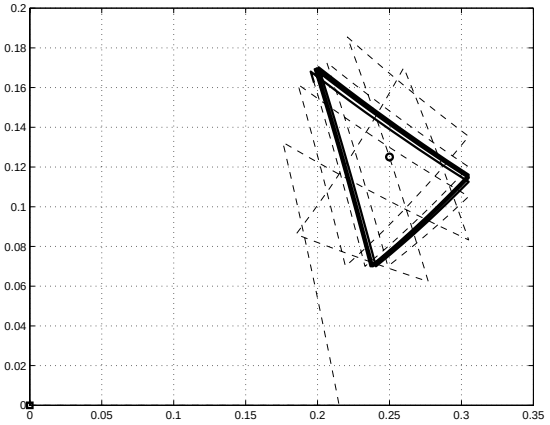

Figure 11. Phase portrait of the trajectories in the furnace example.

6. References

Hedlund, S. and A. Rantzer (1999): "Optimal control of hybrid systems." In *IEEE Conference on Decision and Control*. Phoenix.