CDP Tool, a Matlab Tool for Optimal Control of Hybrid Systems

Hedlund, Sven; Rantzer, Anders

1999

CDP Tool

A MATLAB Tool for Optimal Control of Hybrid Systems

Sven Hedlund
Anders Rantzer

Department of Automatic Control
Lund Institute of Technology
December 1999

1. Introduction

This manual describes CDP Tool, a MATLAB tool that solves hybrid optimal control problems via CDP (Convex Dynamic Programming). The manual 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[HR99]. Section 4 gives a complete description of the MATLAB commands of this tool, while Section 5 demonstrates the usage in some examples.

1.1 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. 3.4 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

Table 1 Commands for optimal control of hybrid systems.

command	description
cdplows	Compute a lower bound on the value function, single-point maximization
cdplowes	Compute a lower bound on the value function of an exponential time weighting problem, single-point maximization
cdplowm	Compute a lower bound on the value function, multi-point maximization
cdplowem	Compute a lower bound on the value function of an exponential time weighting problem, multi-point maximization
cdpctrl	Compute a control signal, based on an approximation of the value function
cdpsim	Simulate controlled system
cdpsimf	Simulate controlled system, fixed time step
cdpsime	Simulate controlled system with exponential time cost function
cdpsimef	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
cdpsf	"cdp simulation file", used by cdpsim
cdpsfe	"cdp simulation file", used by cdpsime

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.

3.1 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] [HR99]. 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.

3.2 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.

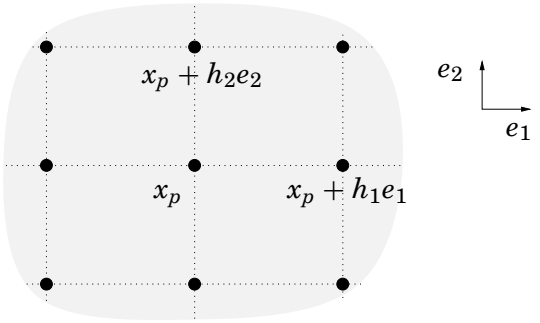

Figure 1 Illustration of the discretization grid in $\mathbf{R}^2$.

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. The value function commands can be set to use a method (that is presented in [HR99]) 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.

[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.

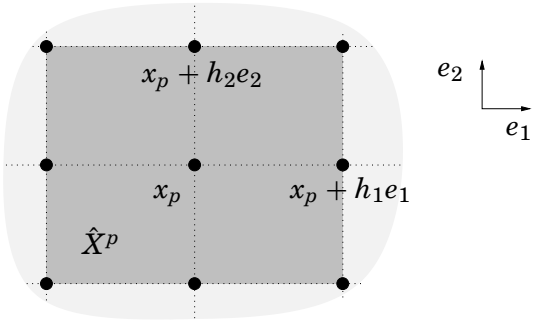

Figure 2 Illustration of $\hat{X}^p$ in $\mathbf{R}^2$.

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 $\Omega_u = \{-1, 1\}$.

3.3 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 `cdplowm` and `cdplowem` 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 function (2) subject to the dynamics in (1), where one of the constraints is on the continuous state:

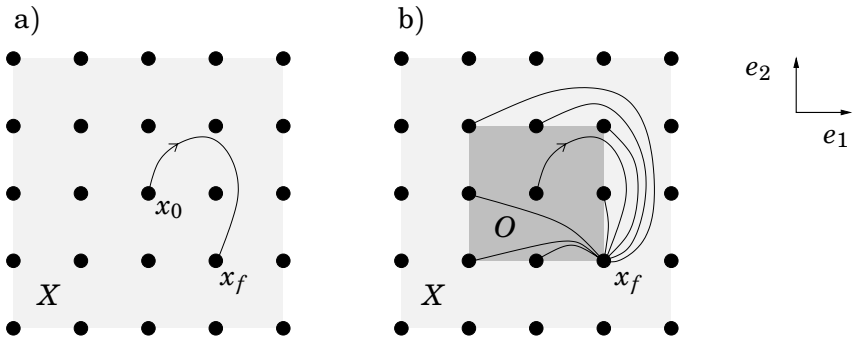

Figure 3 Various choices of maximization in $\mathbf{R}^2$ with corresponding optimal state trajectories. a) Single point maximization. b) Region maximization.

$x(t) \in X$. 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, `cdplowm` and `cdplowem`, 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.

3.4 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

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

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 `cdplowes` and `cdplowem` implement a discretized version of (13). The former function perform a single-point maximization, the latter one a multi-point maximization.

3.5 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 `cdpctrl`, that uses (14) to compute the control signal in a mesh of points, will in most practical cases take the result from `cdplowm` or `cdplowem` as input.

3.6 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 `cdpsim` and `cdpsime`, but there also exist faster, less accurate fixed time step size versions, `cdpsimf` and `cdpsimef`.

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 `cdpsf` and `cdpsfe` 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.

cdpctrl

Purpose

Derive a feedback control law from a value function approximation.

Synopsis

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

Description

`cdpctrl` derives a feedback control law from a value function approximation returned by `cdplowm` or `cdplowem`.

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. 3.2). 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 $V_j - V_i + (1-$`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

`cdplowm, cdplowem`

cdplowem & cdplowes

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] = cdplowem(f, l, s, a, uv, O, XQ, N, tb)
[V, xv, W] = cdplowes(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). `cdplowes` computes the cost for bringing (1) from (x_0, q_0) to (x_f, q_f), while `cdplowem` computes the value function for a region, $\{(x,q) : x \in O, q \in Q\}$[2]. The commands can be forced to bound the value function from below.

Parameters

The parameters that are not found below are described under `cdplowm` and `cdplows`

`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 `cdplow`-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 `cdplowm` and `cdplows`. The difference is that the matrix `V` that is returned by `cdplowem` and `cdplowes` corresponds to $\tilde{V}_q$ in (13).

See Also

`cdplows, cdplowm, linprog`

[2] Cf. Section 3.3 for further details.

cdplowm & cdplows

Purpose

Compute a lower bound on the value function

Synopsis

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

Description

Both of the two commands, in the sequel referred to as the cdplow-commands, compute an approximation of the value function implied by (2). `cdplows` computes the cost for bringing (1) from (x_0, q_0) to (x_f, q_f), while `cdplowm` computes the value function for a region, $\{(x,q) : x \in O, q \in Q\}$[3]. 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 cdplow-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 3.2), such that `h(k)` is the distance between the grid points in the e_k-direction.

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 `cdpctrl` 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 $\min_{x \in \hat{X}^p} l_q(x_p, q, u)$.

[3] Cf. Section 3.3 for further details.

`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. 3.2). Use an empty vector (`uv = []`) as a place holder if there is no continuous input.

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

`O` specifies the region of maximization, $O \subseteq X$, for the multi-point maximizing function `cdplowm` (cf. Section 3.3). `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

`cdplowes, cdplowem, linprog`

cdpsim & cdpsimf

Purpose

Simulate a controlled hybrid system.

Synopsis

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

Description

`cdpsim` and `cdpsimf` simulate a hybrid system using the feedback control law returned by `cdpctrl`. The difference between the commands is that `cdpsim` calls an ODE solver in MATLAB, while `cdpsimf` 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 `cdpctrl` that gives the control law for u.

`Q` is an $(n+1)$-dimensional matrix returned by `cdpctrl` 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 `cdpctrl`.

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

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

`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. 3.2). 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

`cdpctrl, cdpsime, cdpsimef`

cdpsime & cdpsimef

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

Description

`cdpsime` and `cdpsimef` simulate a controlled hybrid system, for which the control law returned by `cdpctrl` has been derived from a cost function on the form (11). The difference between the commands is that `cdpsime` calls an ODE solver in MATLAB, while `cdpsimef` 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 `cdpsim` and `cdpsimf`.

`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 `cdpctrl` that gives the control law for u.

`Q` is an $(n+1)$-dimensional matrix returned by `cdpctrl` 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 `cdpctrl`.

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

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

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

See Also

`cdpctrl, cdpsim, cdpsimf`

crop

Purpose

To crop a multi dimensional array.

Synopsis

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

Description

Having computed a value function, `V`, using `cdplowm` or `cdplowem`, 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 3.3), the value function that `cdplowm` and `cdplowem` 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

`cdplowm`, `cdplowem`

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 (`cdplowm`, `cdplows`, `cdplowem`, and `cdplowes`) to solve linear programs. The default file `linprog.m` forwards the LP solving request to the MATLAB program `PCx`.[4] 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

`cdplowm, cdplows, cdplowem, cdplowes`

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

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.

5.1 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.

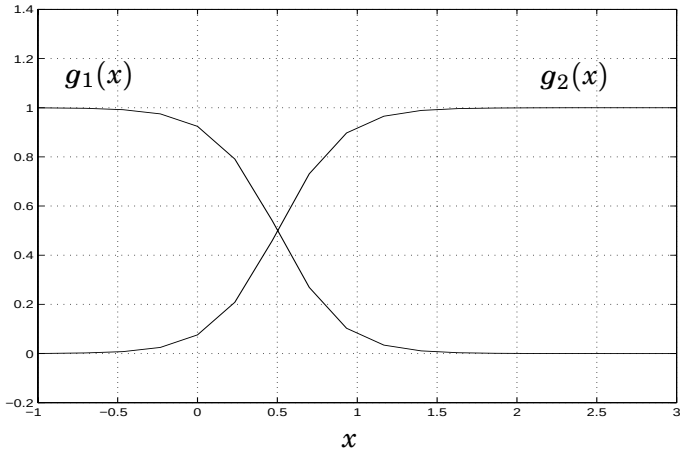

Figure 4 Gear efficiency at various speeds.

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 `cdplowm` 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 grid point
>> xmin = xmin + odx;
>> xmax = xmax + odx;
>> Q    = 2;
>> XQ   = [xmin; xmax; Q];

>> [V,xv]   = cdplowm('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] = cdpctrl('car_f','car_l','car_s',uv,Vc,xvc);

>> x0   = [-5;0];
>> q0   = 1;
>> xq0  = [x0; q0];
>> tend = 8;
>> [tv,xv2,qv] = cdpsim('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');
```

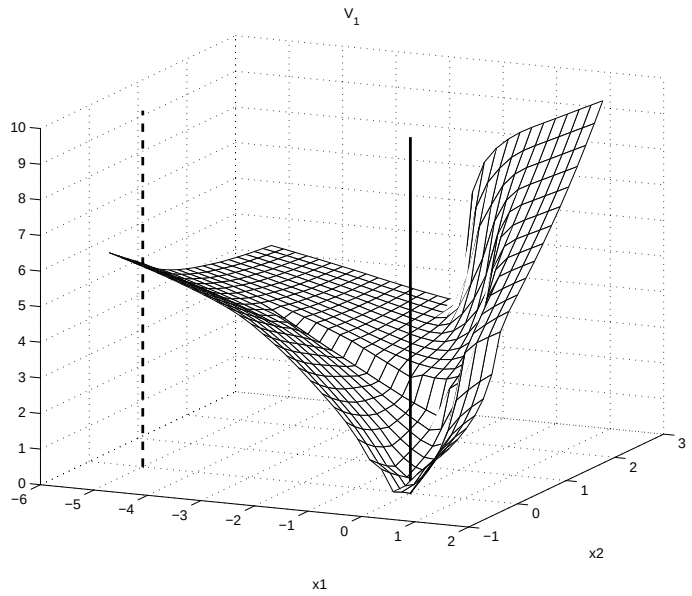

Figure 5 Plot of V_1. The initial point, x_i, is marked with a vertical dashed line, the final point, x_f, with a solid line.

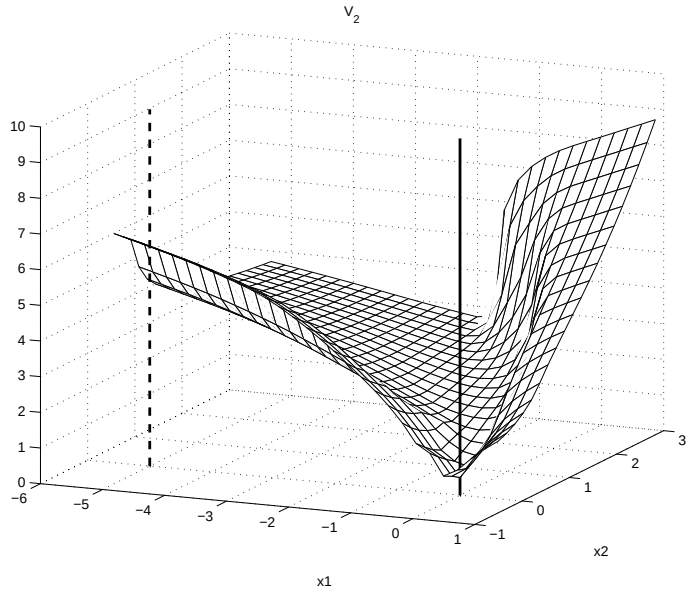

Figure 6 Plot of V_2.

```
>> 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.

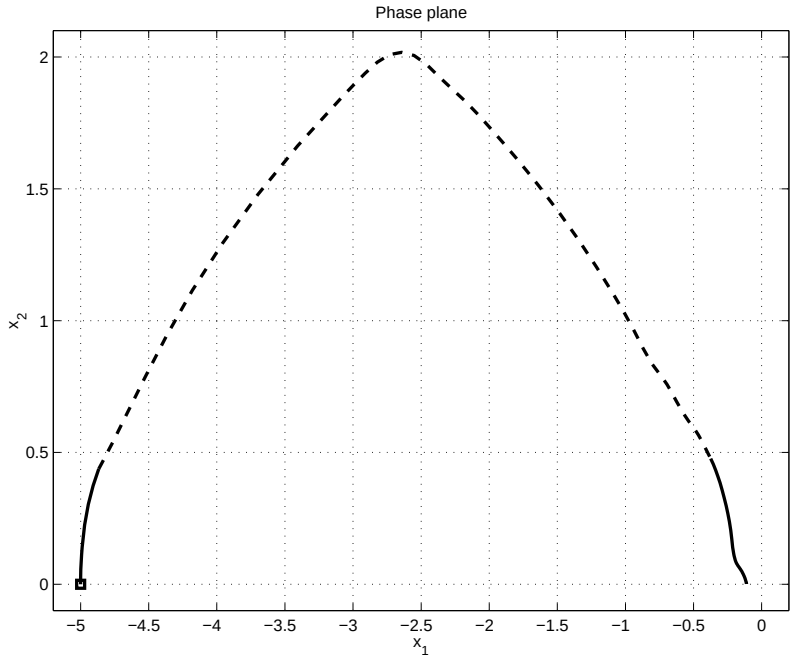

Figure 7 Phase portrait of a simulation. The solid line shows where gear number one has been used, the dashed line shows the second gear. The initial point is marked with a square.

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] = cdplows('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');
>> 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.

5.2 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 time. This is handled by

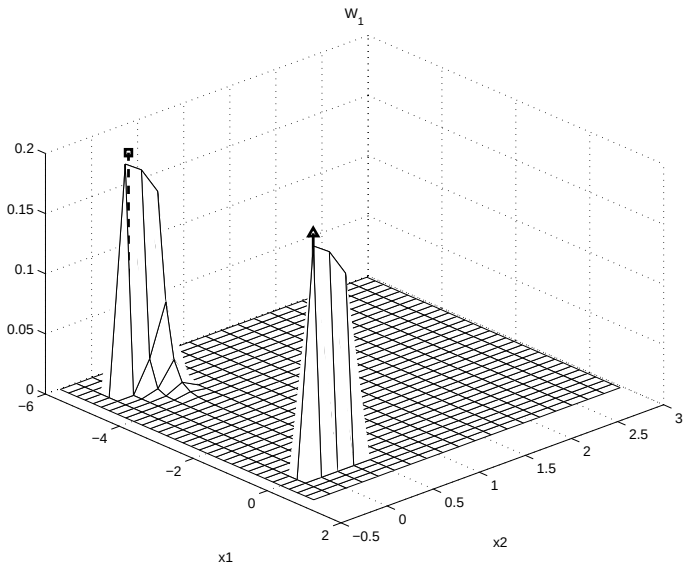

Figure 8 Plot of W_1. The initial point, x_i, is marked with a vertical dashed line, the final point, x_f, with a solid line.

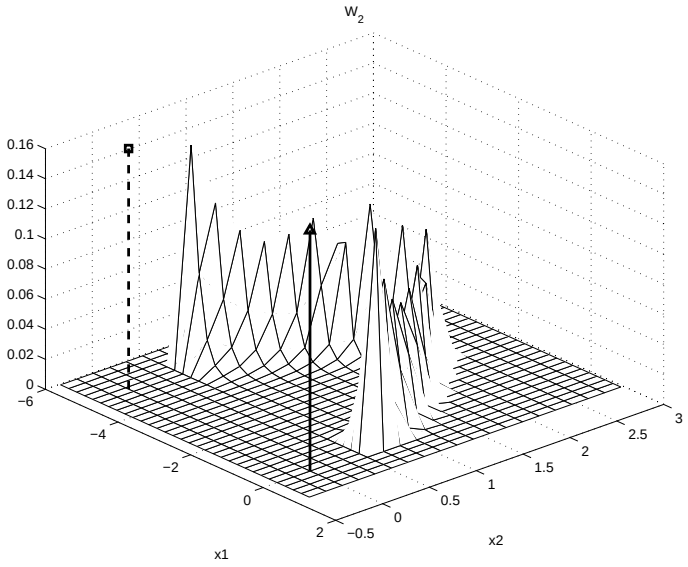

Figure 9 Plot of W_2.

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

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

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 `cdplowem`. 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]   = cdplowem('fu_f','fu_l','fu_s',1,uv,0,XQ,N,0);
>> [Vc, xvc] = crop(V,xv,0);
```

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

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

>> x0   = [0; 0];
>> q0   = 3;
>> xq0  = [x0; q0];
>> tend = 6;
>> [tv,xv2,qv] = cdpsime('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.

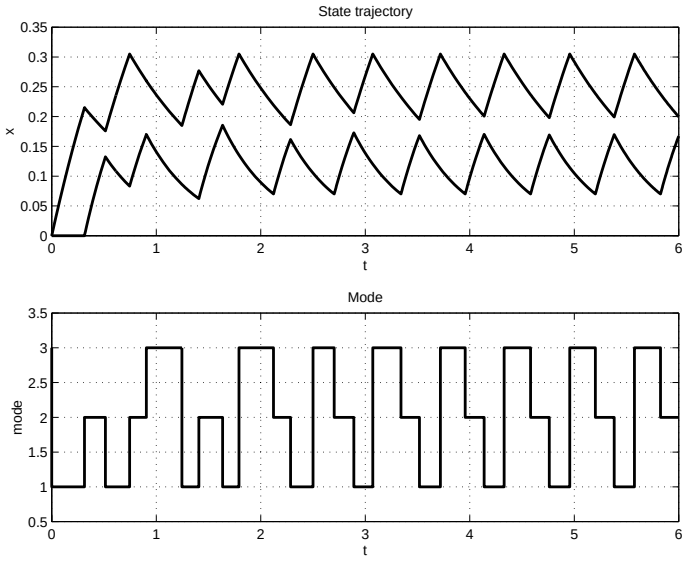

Figure 10 Time plot of the trajectories in the furnace example.

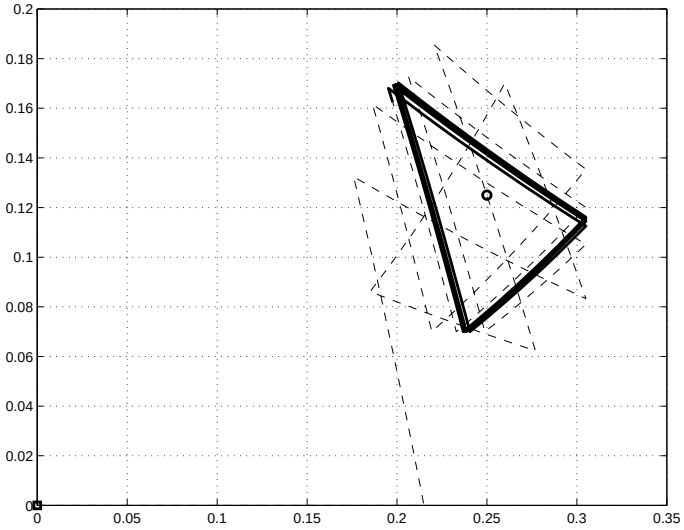

Figure 11 Phase portrait of the trajectories in the furnace example.

6. References

[HR99] S. Hedlund and A. Rantzer. Optimal control of hybrid systems. In *IEEE Conference on Decision and Control*, Phoenix, 1999.