# Software user-guide

This brief appendix aims to present some of the software packages, implemented during this doctoral period at the University of Cagliari. This software was used to obtain part of the results described in this thesis.

The software is implemented in MATLAB, version 6.0, Release 12. It can be downloaded from

*http://www.diee.unica.it/∼dcorona/thesis.html*.

With the package *STP2_corona.zip* you can:

1. Construct the *switching tables* for a $\mathbb{R}^2$ hybrid automata, modelled in this thesis, that provide the feedback switching law of the considered optimal control and stability problem (Chapters 3,4,7) – Function ***regions.m***.
2. Visualize graphically these tables – Function ***plot_tables.m***.
3. Obtain the number of switches $\bar{N}$ (Chapters 6,7) where the convergence is achieved. The convergence criterion has not been automatized, hence this decision is up to the user, by direct visualization of the tables.
4. Simulate the use of the switching tables, in both cases of $N$ *finite* and *infinite* – Function ***simulation.m***.
5. In addition an efficient general function that calculates the LQR cost from a given initial point and for a given time interval is proposed – Function ***index.m***.

Additionally with the package *STP4_corona.zip* you can implement the STP in $\mathbb{R}^4$. We decided to provide separate files because of the more complex data structure in $\mathbb{R}^4$ compared to $\mathbb{R}^2$. The STP4 software guide is in Section 0.5.

## 0.1 Function *regions.m*

This software is for two dimensional use, i.e., the state space $\boldsymbol{x} \in \mathbb{R}^2$. This implies that matrices $\boldsymbol{A}$, $\boldsymbol{Q}$ and $Jump$ are matrices of class $\mathbb{R}^2$.

The function receives in input the following data:

1. The *hybrid automata* (dynamics, jumps, edges and minimum permanence time);
2. The *optimal control problem* (weight matrices and number of available switches);
3. The discretization data (Time and space discretization);
4. Eventually, to be used when you need to keep increasing $N$ to meet convergence, the previously calculated table.

Before proceeding further with the help of the software we provide the notion of MATLAB *matrix array*.

A matrix array is an array whose elements are matrices. A set of matrices $\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3$ can be collected in a unique data structure

$>>$ $\boldsymbol{A}$= $\{\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3\}$

Each element is recalled with the typing of the required index within brackets $\{\}$. For example, the command

$>>$ $\boldsymbol{A}\{1\}$

shows matrix elements of $\boldsymbol{A}_1$. In general also matrices of matrices can be represented. For example

$>>$ $\boldsymbol{M} = \{\boldsymbol{M}_{1,1}, \boldsymbol{M}_{1,2}; \boldsymbol{M}_{2,1}, \boldsymbol{M}_{2,2}\}$

where $\boldsymbol{M}_{i,j}$, $i, j = 1, 2$ are matrices. The command

$>>$ $\boldsymbol{M}\{2, 2\}$

shows matrix $\boldsymbol{M}_{2,2}$.

### 0.1.1 Initial use: no tables are calculated

At the MATLAB prompt type:

$>>$ $Table = regions(A, Q, G, Jump, d\_min, N_\vartheta, \tau_M, N_t, N)$

where

**INPUT**

1. $\boldsymbol{A}$ is a *matrix array* $1 \times s$, $s$ is the number of locations, that contains all dynamics of the automaton. Note that the software, in this preliminary version, *does not have internal checks* on the stability of each element, hence make sure that at least one matrix of the array $\boldsymbol{A}$ is stable.
2. $\boldsymbol{Q}$ is a *matrix array* $1 \times s$, that contains all weights in the LQR cost. Note that *all* Q must be *non negative* definite.
3. $G$ is a matrix $s \times s$ of edges. If there exists an arc from location $i$ to location $j$ then set the element $G(i, j) = j$, else set $G(i, j) = 0$. Note that in this context there are no self-loops, hence set $G(i, i) = 0$.
4. $Jump$ is a *matrix array* $s \times s$ of *switching jump matrices* $Jump\{i, j\}$. In case of continuous state set $Jump\{i, j\} = eye(2)$.
5. $d\_min$ is a row vector that contains the minimum permanence times in each location. Note that each element of this vector must be positive or null.
6. $N_\vartheta$ is the number of *equally* spaced points on the unitary semisphere. Typical values are 51, 71, 101, 151. You might prefer odd values in order to represent the point on the $x_2$ axis. Point 1 corresponds to $[1, 0]'$, point $\frac{N_\vartheta+1}{2}$ corresponds to $[0, 1]'$, point $N_\vartheta$ to $[-1, 0]'$.
7. $\tau_M$ maximum time exploration, in theory infinite. Note that an appropriate value should be 4 or 5 times the slowest time constants of the stable matrices $\boldsymbol{A}_i$

8. $N_t$ number of points in the time exploration. Note that this number should be coordinated with $\tau_M$ in order to obtain a fine enough time step $dt = \frac{\tau_M}{N_t}$. Unless $\boldsymbol{A}$'s have high frequencies modes, values between $dt = 10^{-2}, 10^{-3}$ are acceptable.
9. $N$ number of allowed switches.

**Remark 0.1 (Stabilization usage)** *If you are aiming to use the software to design a table that* stabilizes *a switched system where all dynamics are unstable, make sure the following:*

1. *Insert in the matrix array $\boldsymbol{A}$ a stable dynamics $\boldsymbol{A}\{s+1\}$, typically, a good choice is to take one of the unstable dynamics (with all positive real parts eigenvalues) with opposite sign. Rotating stable dynamics are observed to behave better.*
2. *Insert in the matrix array $\boldsymbol{Q}$ an extremely expensive positive definite matrix. Good examples*[1] *are*

   $>>\ \boldsymbol{Q}\{s+1\} = 1e10 * eye(2).$

3. *The matrix $G$ must be complete, i.e., all its terms out of diagonal are non null. If a switched system (augmented) of three locations is considered then*

$$G = \begin{bmatrix} 0 & 2 & 3 \\ 1 & 0 & 3 \\ 1 & 2 & 0 \end{bmatrix}.$$

4. *All Jump matrices are the identity.*

■

**OUTPUT**

The calculations may be long, hence a sequence of dots are visualized to confirm that the software is effectively running. Each *carriage return* in the line of dots indicates that a new location $i$ is being processed, hence the table $\mathcal{C}_k^i$ is in progress of construction.

In the end the lines of dots will be $(N+1)\times s$, where $s$ is the number of locations.

The data, residual cost and color from each point of the unitary semisphere and for each dynamics, is collected in a matrix array $Table$. The data structure requires further explanations.

- $Table\{k\}$, $k = 1, \ldots, N+1$, contains the information relative to $k-1$ remaining switches. For example $Table\{3\}$ contains the residual cost and the color from each point of the semisphere, when 2 switches are left. Hence $Table\{N+1\}$ is the last calculated one, for $N$ available switches. The command

  $>>\ Table$

  lists this matrix array.
- Each element of $Table$, $Table\{k\}$, is a matrix of $s$ rows and $2N_\vartheta$ columns. In other words each row $i$, i.e., the current dynamics, is a vector of $2N_\vartheta$ elements. This should be seen as a list of $N_\vartheta$ *couples*, (a couple per point of the discretization) where the *odd* element is the residual cost, and the even *element* is an integer $j = 1, \ldots, s$ that indicates the switching strategy.

[1] MATLAB exponential number notation.

**Example 0.1** *Assume that* $N_\vartheta = 101$, $N = 10$ *and* $s = 4$. *Then the element*

$$>> \ Table\{k\}(i, 2h-1)$$

*contains the residual cost when* $k-1$ *switches are left, from location* $i$ *and from point indexed* $h = 1, \ldots, N_\vartheta$ *on the semisphere, and*

$$>> \ Table\{k\}(i, 2h)$$

*contains the location index where it is optimal to switch.* ■

**Remark 0.2** *It is a good habit, when the function* **region.m** *has terminated, to save the data by running the MATLAB command*

$$>> \ save \ < file\_name > \ Table, A, Q, G, Jump, d\_min, N_\vartheta, \tau_M, N_t, N$$

*that stores the input and the calculated data in a file called* $< file\_name >$. *To load this file type the MATLAB command*

$$>> \ load \ < file\_name >$$

*from the belonging directory.*

■

### 0.1.2 Iterative use: a set of tables are available

Use this modality when you want to keep calculating tables for increasing values of $N$. More specifically: the program has constructed already $N$ tables, $Tab$. Probably you want to calculate $M = N + \tilde{N}$, without losing the previous effort.

Hence, at the MATLAB prompt type:

$$>> \ Table = regions(A, Q, G, Jump, d\_min, N_\vartheta, \tau_M, N_t, M, Tab)$$

where:

**INPUT**

All elements *must* be the same as in the previous section, except the last two.

1. $M$: new number of allowed switches, greater then the previous one.
2. $Tab$ is the set of tables previously calculated.

**OUTPUT**

See previous section.

## 0.2 Function *plot_tables.m*

This function prepares the data to plot the switching tables in $\mathbb{R}^2$.

We assume that the program region has been executed and a set of tables has been calculated. At the MATLAB prompt type

$>>\ [X, Y, T] = plot\_table(Table);$

The data stored in $X, Y, T$ contains the information for the input of the MATLAB built-in function ***pcolor***.

If you want to visualize the table $\mathcal{C}^i_{k-1}$ ($k-1$ switches are left from location $i$) then type

$>>\ pcolor([X\{k\}; 0\ 0; 0\ 0], [Y\{k\}; 0\ 0; 0\ 0], [T\{k, i\}; 2\ 2; 1\ 1])$

if the automaton has 2 locations,

$>>\ pcolor([X\{k\}; 0\ 0; 0\ 0; 0\ 0], [Y\{k\}; 0\ 0; 0\ 0; 0\ 0], [T\{k, i\}; 3\ 3; 2\ 2; 1\ 1])$

if the automaton has 3 locations,

$>>\ pcolor([X\{k\}; 0\ 0; \ldots; 0\ 0], [Y\{k\}; 0\ 0; \ldots; 0\ 0], [T\{k, i\}; s\ s; \ldots; 1\ 1])$

if the automaton has $s$ locations.

**Remark 0.3 (Color mapping)** *The color associated to location $i$ is the color of table $Table\{1\}(i, :)$ and you may visualize it with command*

$>>\ pcolor([X\{1\}; 0\ 0; \ldots; 0\ 0], [Y\{1\}; 0\ 0; \ldots; 0\ 0], [T\{1, i\}; s\ s; \ldots; 1\ 1]).$

*This command produces a disk with the color associated to the $i-th$ location.* ■

## 0.3 Function *simulation.m*

From a given initial hybrid state $(\boldsymbol{x}, i)$ it is possible to use the tables, calculated by function ***regions.m***, to calculate the optimal switching intervals, the optimal switching sequence, the optimal cost and the optimal trajectory.

At the MATLAB prompt type

$>>\ [T, I, J, X] = simulation(A, Q, Jump, d\_min, \tau_M, N_t, \boldsymbol{x}, i, Table, op, th);$

where all input variables have been defined in Section 0.1.1, except for

1. $Table$ is the output of program ***regions.m***;
2. $op$, a parameter so defined:
   - $op = 0$, *finite* number of switches, uses all tables;
   - $op = 1$, *infinite* number of switches, uses only the last calculated tables.
3. $th$ is a terminating criterion on the norm of the continuous state $\boldsymbol{x}$, usually values $th = 10^{-3}, 10^{-4}$ are acceptable.

**OUTPUT**

The subroutine ***simulation.m*** outputs the following data:

1. $T$ is an array of the permanence time in each location;

2. $I$ is an array of the visited locations during the evolution; $I(k)$ is the index of the location when $k$ switches are available, while $T(k)$ is the time interval spent in location $I(k)$.
3. $J$ is the total cost of the evolution;
4. $X$ is a sequence of points $\boldsymbol{x}$ that describes the evolution.
   To sketch the plot of the evolution type the command

   $>>\ plot(X(1,:), X(2,:))$

**Example 0.2** *In Section* **??** *we implemented the switched system model of a servomechanism with gear box. We firstly run the function* **regions.m** *with the given numerical values, hence we simulated an evolution from the therein given initial point. The function* **simulation.m** *exited the following numerical values:*

1. $T = [0.20, 0.20, 1.07, 2.53, 0.20]$,
2. $I = [1, 3, 5, 6, 5, 3]$,
3. $J = 4.75$.

*Note that the array $T$, the switching intervals, has been converted into $\mathcal{T}^*$ that expresses the switching instants in an absolute time scale.*

*In addition vector $X$ has been used to sketch the evolution depicted in Figure* **??**.

■

## 0.4 Function *index.m*

This function serves to calculate the integral

$$J = \int_0^{\varrho} \boldsymbol{x}'(t)\boldsymbol{Q}\boldsymbol{x}(t)dt, \tag{0.1}$$

subject to $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t)$ and $\boldsymbol{x}(0) = \boldsymbol{x}_0$.

Albeit not directly involved in the STP its usage is crucial for the described functions. Moreover it is quite general, hence we decided to describe it better.

In this paragraph $\boldsymbol{A}$, $\boldsymbol{Q}$ are general[2] square matrices.

To calculate the value of the integral (0.1), we preliminary need to solve the Lyapunov matrix equation. To this aim, at the MATLAB prompt, type

$>>\ [\boldsymbol{Z}, flag] = lyap\_mod(\boldsymbol{A}', \boldsymbol{Q})$

that solves the Lyapunov matrix equation $\boldsymbol{A}'\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{A} = -\boldsymbol{Q}$ and returns a $flag$ whose value is

- $-1$ if $\boldsymbol{Z}$ does not exists or it is not unique;
- $0$ if $\boldsymbol{Z}$ exists and the matrix $\boldsymbol{A}$ is Hurwitz;
- $1$ if $\boldsymbol{Z}$ exists and the matrix $\boldsymbol{A}$ is non Hurwitz.

The $flag$ variable is needed because when it assumes the values $0, 1$ it is possible to solve the integral *analytically*, so gaining in precision and computational time. In fact, as described in Appendix **??**, it holds

[2] Except for $\boldsymbol{Q} \geq \boldsymbol{0}$.

$$J = \int_0^{\varrho} \boldsymbol{x}'(t)\boldsymbol{Q}\boldsymbol{x}(t)dt = \boldsymbol{x}_0'(\boldsymbol{Z} - \bar{\boldsymbol{A}}'(\varrho)\boldsymbol{Z}\bar{\boldsymbol{A}}(\varrho))\boldsymbol{x}_0.$$

In case $flag = -1$ then the Lyapunov equation has non unique or non existing solution, hence the cost must be calculated *numerically*. To this purpose we found satisfactory the implementation of a *constant step trapezoidal method* [**?**].

Now type

$>>\ J = index(\boldsymbol{A}, \boldsymbol{Q}, \boldsymbol{x}_0, \boldsymbol{Z}, flag, \varrho, dt)$

where $dt$ is an appropriately chosen time interval, useful if the integral is calculated numerically.

Note that the call to function ***lyap_mod.m*** may be done inside the function ***index.m***, but in this case the computational time of ***index.m*** would increase. This is undesirable, because this function is called by ***regions.m*** at least $Ns^2N_\vartheta N_t$ times.

Similarly the execution of

$>>\ J = index(\boldsymbol{A}, \boldsymbol{Q}, \boldsymbol{x}_0, \boldsymbol{Z}, flag)$

calculates

$$J = \int_0^{\infty} \boldsymbol{x}'(t)\boldsymbol{Q}\boldsymbol{x}(t)dt, \tag{0.2}$$

subject to $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t)$ and $\boldsymbol{x}(0) = \boldsymbol{x}_0$. In this case the computation is simplified. In fact it holds:

1. if $flag = 0$ then $J = \boldsymbol{x}_0'\boldsymbol{Z}\boldsymbol{x}_0$;
2. if $flag = -1, 1$ then, immediately, $J = +Inf$.

## 0.5 Function *regions4.m*

Download the file *STP4_corona.zip* from

*http://www.diee.unica.it/∼dcorona/thesis.html*.

The function *main* is called ***regions4.m***. It implements the STP in the fourth dimensional case. Hence all dynamics and weight matrices are of class $\mathbb{R}^{4\times4}$. For this case we neglected the state jumps.

At the MATLAB prompt type

$>>\ [Table, X] = regions4(A, Q, G, d\_min, N_\xi, N_\varphi, N_\vartheta, N)$

where the input data $A, Q, G, d\_min, N$ have already been described in Section 0.1.1.

The input variables $N_\xi, N_\varphi, N_\vartheta$ represent the discretization of the unitary semisphere in $\mathbb{R}^4$. To have a better interpretation of these values see also **??**.

An appropriate choice of these values should be $N_\vartheta = 2N_\varphi = 4N_\xi$, as it has been motivated in Appendix **??**. In the example implemented in Section **??** we chose $N_\xi = 15$. This choice leads to a discretization of $8581$ points, sparse in $\Sigma_4$, that was considered acceptable.

**OUTPUT**

The calculations are long, hence a sequence of dots are visualized to confirm that the software is effectively running.

The data, residual cost and switching strategy, from each point of the unitary semisphere and for each dynamics, is collected in a matrix array $Table$, whose structure requires further explanations.

It is a matrix array $Table\{k\}$, $k = 1, ..., N+1$.

- $Table\{k\}$, $k = 1, \ldots, N+1$, contains the information relative to $k-1$ remaining switches. For example $Table\{3\}$ contains the residual cost and the color from each point of the semisphere, when 2 switches are left. Hence $Table\{N+1\}$ is the last calculated one, for $N$ available switches. The command

  $$>> \ Table$$

  lists this matrix array.
- The element $Table\{k+1\}(i, h, 1)$ is the residual cost from point indexed by $h$ (semisphere, see $X$) and from location $i$.
- The element $Table\{k+1\}(i, h, 2)$ is the color mapping of point indexed by $h$ (semisphere, see $X$) and from location $i$.
- The data structure $X$ is a matrix whose rows represent the polar angles of the unitary semisphere in $\mathbb{R}^4$. Hence $X(h, :)$ is a point in $\mathbb{R}^4$ in polar coordinates and $\varrho = 1$.

**Remark 0.4** *It is a good habit, when the function* **region4.m** *has terminated, to save the data by running the MATLAB command*

$$>> \ save \ < file\_name > \ Table, X, A, Q, G, d\_min, N_\xi, N$$

*that stores the input and the calculated data in a file called* $< file\_name >$*. To load this file type the MATLAB command*

$$>> \ load \ < file\_name >$$

*from the belonging directory.* ■

## 0.6 Function *simulation4.m*

From a given initial hybrid state $(\boldsymbol{x}, i)$ it is possible to use the tables, calculated by function ***regions4.m***, to calculate the optimal switching intervals, the optimal switching sequence, the optimal cost and the optimal trajectory.

At the MATLAB prompt type

$$>> \ [T, I, J, X1] = simulation4(\boldsymbol{x}, i, d\_min, A, Q, Table, X, Ncsi, dt, op, th)$$

where all input variables have been defined in Section 0.1.1 and 0.5, except for

1. $Table, X$ are the output of program ***regions4.m***;
2. $dt$ represents the time step of the simulation, usually $dt = 10^{-3}, 10^{-4}$ are efficiently fine;
3. $op$, a parameter so defined:

- $op = 0$, *finite* number of switches, uses all tables;
- $op = 1$, *infinite* number of switches, uses only the last calculated tables.

4. $th$ is a terminating criterion on the norm of the continuous state $\boldsymbol{x}$, usually values $th = 10^{-3}, 10^{-4}$ are acceptable.

**OUTPUT**

The subroutine ***simulation4.m*** outputs the following data:

1. $T$ is an array of the permanence time in each location;
2. $I$ is an array of the visited locations during the evolution; $I(k)$ is the index of the location when $k$ switches are available, while $T(k)$ is the time interval spent in location $I(k)$.
3. $J$ is the total cost of the evolution;
4. $X1$ is a sequence of points $\boldsymbol{x}$ that describes the evolution.

In this case the trajectory of the evolution has no geometrical interpretation. However it is possible to sketch each row of matrix $X1$, i.e., $X1(i,:)$, $i = 1, 2, 3, 4$ that represents the time evolution $x_i(t)$, with time step $dt$.