SVM viability kernel approximation and resilience computation software Version 1.0

User Guide

Laetitia Chapel

December 20, 2007

1 Introduction

We developed software to approximate viability kernels and compute resilience values using Support Vector Machines (SVMs). In this version:

- we use a regular grid (an active learning algorithm will come in a forthcoming version);
- the capture basins and resilience values are computed in dimension $d + 1$ (the computation in dimension d will come in a forthcoming version);
- we can define heavy controllers.

The software is written in the Java programming language. The software has two modes: a GUI mode and a batch mode.

2 Glossary

- **Viability kernel (Viab(K)):** set of all the points x for which there exists at least one control function $t \rightarrow u(t)$ such that the whole evolution starting from x always remains in the viability constraint set K.
- **Capture basin (Capt(K,C)):** set of states for which there exists at least one trajectory that reaches a target C in finite time, without leaving K.
- **Resilience values:** computed as the inverse of restauration cost. Inside the viability kernel; the cost is null, if there exists a trajectory that allows the system to come back to K, the cost is finite and its value is a function of the time needed to restore the system; if there is no control function that allows the restauration of the system, the cost is infinite.
- **Heavy controller:** keep the control constant as long as the system stays inside the viability kernel and change it only when the system reaches the boundary on the kernel, by choosing the first control that keeps the system inside $Viab(K)$.
- **SVMs:** classification procedures that define a separating hypersurface between examples x_i, associated with their label y_i. SVM training provides function:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i k(x, x_i) + b.$$

The points x_i for which $\alpha_i > 0$ are called support vectors. The sign of the function $f(x)$ gives the label of the point x. $k(x, x_i)$ is called kernel, for example the gaussian kernel has the following expression:

$$k(x, x_i) = \exp\left(-\gamma \parallel x - x_i \parallel^2\right), \tag{1}$$

where $\gamma > 0$ is a parameter.

Contents

3 Installation

3.1 Prerequisites

To run the java program, you need to install first:

- Java virtual machine (Sun's JRE environnement 5 or later compulsory), downloadable at
  `http://java.sun.com/javase/downloads/index.jsp`

The program can be run on Windows (Linux ?). The software version 1.0 is provided as

- a .jar file (**SVMVC v1.0.jar**), for users who only want to test the models already implemented in the version 1.0;
- a .zip file (**SVMVC v1.0.zip**), for users who want to implement their own models on the software.

3.2 Installation

The file **SVMVC v1.0.jar** is a java executable file. It runs the software with the Graphical User Interface.

For users who want to implement their own model, unzip the file **SVMVC v1.0.zip** to a new directory named **SVMVC v1.0**. The directory now contains

<bin>	Subdirectory	contains the binary files of the program
<dist>	Subdirectory	contains the **SVMVC v1.0.jar** file
<doc>	Subdirectory	contains the javadoc for package *model*
<lib>	Subdirectory	contains some libraries used by the software
<src>	Subdirectory	contains the source code
*.simu	Files	text files that can be used to run the software in batch mode
license	File	license of the program

3.3 Run the program

Double click on the **SVMVC v1.0.jar** file. You should have the GUI represented in figure 1.

For users who wish to implement their own models, a program like Eclipse (downloadable at `http://www.eclipse.org/downloads/`) can be used to modify the source code. The files included in the <lib> subdirectory must also be included on the build path.

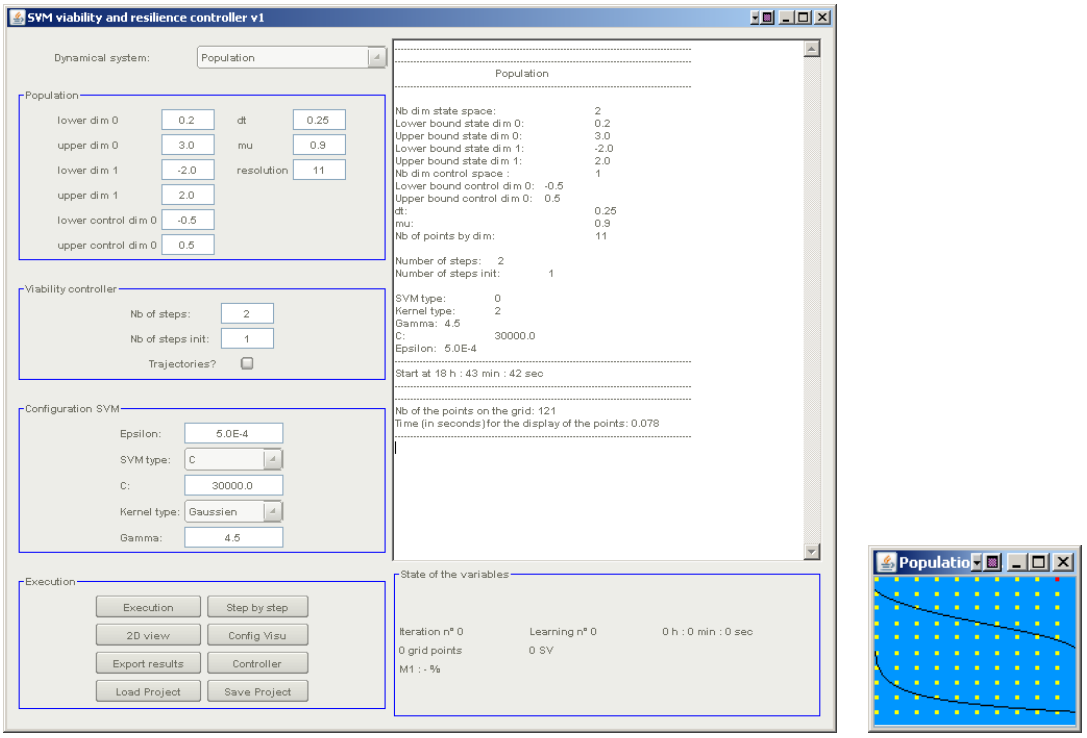

Figure 1: Software in the GUI mode

4 Description of the models

Some models are already implemented in the software. This part gives a description of the dynamics of them, and their viability constraint set.

4.1 Population system [Aubin, 2002]

We consider a simple dynamical system of population growth on a limited space. The state $(x(t), y(t))$ of the system represents the size of a population $x(t)$, which grows or diminishes with the evolution rate $y(t) \in [d; e]$. The size of the population must remain in an interval $K = [a, b]$, with $a > 0$. The inertia bound c limits the derivative of the evolution rate at each time step. The system in discrete time defined by a time interval dt can be written as follows:

$$x(t + dt) = x(t) + x(t)y(t)dt \tag{2}$$
$$y(t + dt) = y(t) + u(t)dt \tag{3}$$

with $-c \leq u(t) \leq +c$. The viability constraint set is the set $K = [a; b] \times [d; e]$. For example, we can use the following parameters: $a = 0.2$, $b = 3$, $c = 0.5$, $d = -2$, $e = 2$.

4.2 Consumption system [Aubin, 1991]

The problem is a consumption problem, defined in 2 dimensions $x(t)$ and $y(t)$. Variable $x(t)$ represents the consumption of a raw material and $y(t)$ its price. The price evolution between two time steps is bounded. The viability constraint set is the set $K = [a; b] \times [d; e]$. The dynamics represent the consumption of the raw material, limited by prices:

$$x(t + dt) = x(t) + (x(t) - y(t))dt \tag{4}$$
$$y(t + dt) = y(t) + u(t)dt, \tag{5}$$

with $-c \leq u(t) \leq +c$. For example, we can use the following parameters: $a = 0$, $b = 2$, $c = 0.5$, $d = 0$, $e = 3$.

4.3 Language system [Abrams and Strogatz, 2003]

We consider the Abrams-Strogatz model of language competition. Two languages A and B are in competition and it is supposed that individuals speak only one language. σ_A (resp σ_B, with $\sigma_B = 1 - \sigma_A$) is the density of speakers of language A (resp B). The language dynamics is given by:

$$\sigma'_A = (1 - \sigma_A)\sigma_A(\sigma_A^{a-1}s - (1 - \sigma_A)^{a-1}(1 - s)), \tag{6}$$

where s is a parameter which is in $[0, 1]$. It measures the social status, the prestige of language A or the politics in favor of language A. The prestige of language B is $1 - s$. For instance, if $s = 0$, the prestige of language A is null and the prestige of language B is maximal and if $s = 1$, this is the opposite. a is a parameter that represents the volatility of language A.

We want to examine the conditions in which the coexistence of speakers of A and B is possible. More precisely, we would like to keep the minority language above the density b. We suppose that the prestige s can change and we include it as a state variable. The state space is therefore bidimensionnal (σ_A, s). Moreover, we suppose that the change of prestige during a given time step is limited: $-c \leq s' \leq c$. Therefore, we get the following system:

$$\begin{cases} \sigma'_A = (1-\sigma_A)\sigma_A(\sigma_A^{a-1}s - (1-\sigma_A)^{a-1}(1-s)) \\ -c \leq s' \leq c \\ b \leq \sigma_A \leq 1-b \\ 0 \leq s \leq 1. \end{cases} \tag{7}$$

and $K = [b; 1-b] \times [0; 1]$. For example, we can use the following parameters: $a = 1.31$, $b = 0.2$, $c = 0.1$.

4.4 Lake system [Carpenter et al., 1999]

The system under consideration encompasses a lake and the farming activities in its watershed. The model combines an ecosystem model of phosphorus dynamics and a controlled model for phosphorus input dynamics. Dynamic of phosphorus in water follows the model:

$$\frac{dP(t)}{dt} = -bP(t) + L(t) + r\frac{P(t)^q}{P(t)^q + m^q}, \tag{8}$$

where P is the mass of phosphorus in the lake water, b is the rate of phosphorus elimination at each time step, m is the P mass in the water for which recycling is half of the maximum rate r. Parameter q sets the steepness of the recycling versus P curve when $P \approx m$. $L(t)$ represents the inputs of phosphorus that come from human activities. We suppose that the lake manager can act directly on the time variation through control u, with u bounded because modifications take time:

$$\frac{dL(t)}{dt} = u, \text{ with } u \in [L_{min}, L_{max}]. \tag{9}$$

We also assume that an oligotrophic lake becomes eutrophic when the amount of phosphorus in the water increases over some fixed threshold P_{max}. We suppose that farmers' benefit depends linearly on the inputs of phosphorus. Consequently, profitability is reached when the value of phosphorus inputs is higher than a given threshold L_{min} and lower than the maximal legal value L_{max}.
The system is thus 2-dimensional:

$$\begin{cases} \frac{dP(t)}{dt} = -bP(t) + L(t) + r\frac{P(t)^q}{P(t)^q + m^q}. \\ \frac{dL(t)}{dt} = u, \end{cases} \tag{10}$$

and $K = [L_{min}; L_{max}] \times [0; P_{max}]$. For example, we can use the following parameters: $L_{min} = 0.1$, $L_{max} = 1$, $P_{max} = 0.5$, $c = 0.1$, $q = 8$, $r = 1$, $m = 1$, $b = 0.8$.

4.5 Bilingual system [Minett and Wang, IP]

In this model, we consider the existence of bilingual individuals. We note: σ_A is the density of speakers of language A, σ_B is the density of speakers of language B, σ_{AB} is the density of speakers of language A and B. The language dynamics is given by this model proposed by Minett-Wang (eliminating variable σ_{AB}):

$$\sigma'_A = (1-\sigma_A-\sigma_B)(1-\sigma_B)^a s - \sigma_A \sigma_B^a (1-s) \tag{11}$$
$$\sigma'_B = (1-\sigma_A-\sigma_B)(1-\sigma_A)^a (1-s) - \sigma_B \sigma_A^a s. \tag{12}$$

As in the previous language model, we suppose we want to keep both languages above a threshold density b, and that the absolute value of the prestige derivative cannot be larger than c. So we have the following system:

$$\begin{cases} \sigma'_A = (1-\sigma_A-\sigma_B)(1-\sigma_B)^a s - \sigma_A \sigma_B^a (1-s) \\ \sigma'_B = (1-\sigma_A-\sigma_B)(1-\sigma_A)^a (1-s) - \sigma_B \sigma_A^a s \\ -c \leq s' \leq c \\ b \leq \sigma_A \leq 1-b \\ b \leq \sigma_B \leq 1-b \\ \sigma_A + \sigma_B \geq 1 \\ 0 \leq s \leq 1. \end{cases} \tag{13}$$

The state space is now 3 dimensional (σ_A, σ_B, s), and $K = [b; 1-b] \times [b; 1-b] \times [0; 1]$.

4.6 LanguageResilience system [Bernard et al., 2007]

We are interested in determining if the Abrams-Strogatz language system (without bilinguals) can come back to the viability kernel. We attribute a cost for the system for each time step such that $\sigma_A < b$ or $\sigma_A > 1-b$. This can be expressed as a 3 dimensional viability problem, where the state is (σ_A, s, C), with C representing the cost. The equations ruling the system in discrete time are:

$$\sigma_A(t+dt) = \sigma_A(t) + dt.\sigma'_A(t) \tag{14}$$
$$-c.dt \leq s(t+dt) - s(t) \leq c.dt \tag{15}$$
$$C(t+dt) = \begin{cases} C(t) \text{ if } \sigma_A < b \text{ or } \sigma_A > 1-b \\ C(t) - \lambda dt \text{ otherwise,} \end{cases} \tag{16}$$

with $\lambda > 0$. We also specify a maximal cost C_M. For $\lambda = 1$, the cost represents the time the system is outside $K = [b; 1-b] \times [0; 1]$. The state space is 3 dimensional $H = [0, 1] \times [0, 1] \times [0, C_M]$.

4.7 LakeResilience system [Martin, 2004]

We are interested in determining if the lake system can come back to the viability kernel. We attribute a cost for the system for each time step such that $P > P_{max}$ or $L < L_{min}$.

This can be expressed as a 3 dimensional viability problem, where the state is (L, P, C), with C representing the cost. The equations ruling the system in discrete time are:

$$P(t+dt) = P(t) + dt.P'(t) \tag{17}$$

$$-c.dt \leq L(t+dt) - L(t) \leq c.dt \tag{18}$$

$$C(t+dt) = \begin{cases} C(t) \text{ if } P < P_{max} \text{ and } L > L_{min} \\ C(t) - \lambda(x)dt \text{ otherwise,} \end{cases} \tag{19}$$

with $\lambda > 0$. We use a cost function consisting of two weighted terms, the first term, which corresponds to the ecological cost, is the time spent in an eutrophic state, the second one, which is an economic cost, measures the duration of the period of negative profits weighted by the norm of these negative profits. The function that associates x with the minimal cost over all trajectories starting at x is then defined by:

$$\lambda(x) = \min_{x(.)}(c_1 \int \chi_{P \geq P_{max}} x(\tau)d\tau + c_2 \int (L_{min} - L)\chi_{L \leq L_{min}} x(\tau)d\tau) \tag{20}$$

with $\chi_{P \geq P_{max}}(x) = 1$ if $P \geq P_{max}$, $\chi_{L \leq L_{min}}(x) = 1$ if $L \leq L_{min}$ and 0 otherwise.
The state space is 3 dimensional $H = [0, L_{max}] \times [0, P_M] \times [0, C_M]$, with $P_M > P_{max}$.
The cost function $\lambda(x)$ equals 0 if $x \in K = [L_{min}, L_{max}] \times [0, P_{max}]$.

4.8 ThinTarget system

In the system in 2 dimensions (x, y), we consider the problem of reaching a thin target $C = [0, 0]$. The system must stay in the space $K = [-1; 1] \times [-1; 1]$. The dynamics is defined in discrete time:

$$\begin{cases} x(t+dt) = x(t) + y(t)dt \\ y(t+dt) = y(t) + u(t)dt, \end{cases} \tag{21}$$

with the control $u(t) \in [-1; 1]$.
We approach the capture basin in finite time T of the system (for example, try $T = 1$).

4.9 Zermelo system [Cardaliaguet et al., 1997]

The problem is derived from the famous Zermelo problem. The aim is to drive a boat in a river, such that it can reach an island as quick as possible, without leaving the constraint set. The state (x, y) represents the position of the boat in the river, where the current decreases when it approaches the boundary of the river. The constraint set K represents the river and C the island to reach. At each time step, the captain of the boat can control its acceleration u and its direction θ. The system in discrete time can be defined by:

$$\begin{cases} x(t+dt) = x(t) + (1 - 0.1y(t)^2 + u\cos\theta)dt \\ y(t+dt) = y(t) + (u\sin\theta)dt. \end{cases} \tag{22}$$

For example, we can put $K = [-6; 2] \times [-2; 2]$ and $C = B(0; 0, 44)$, where B is the unit ball in R^2. Controls must remain in a given interval: $u \in [0; 0, 44]$ and $\theta \in [0; 2\pi]$.
The boat must reach the island before time (try $T = 7$).

4.10 CarOnTheHill system [Moore and Atkeson, 1995]

The system is in 2 dimensions: the car position x and its velocity x'. The car can be controlled thanks to a continuous variable in one dimension: the thrust a. The aim is to keep the car in a given constraint set, while reaching the target as fast as possible, with a small velocity: $C \in [0,5;0,7] \times [-0,1;0,1]$. The car must remain in the constraint set $K = [-1;1] \times [-2;2]$. The thrust is limited $a \in [-4;4]$. The evolution of the velocity x' is function of the position x :

$$x'' = \frac{a}{\sqrt{1+(H'(x))^2}} - \frac{gH'(x)}{1+H'(x)^2}, \tag{23}$$

with $g = 9.81$ and :

$$H'(x) = \begin{cases} x^2 + x & \text{if } x < 0 \\ x/\sqrt{1+5x^2} & \text{if } x \geq 0. \end{cases} \tag{24}$$

We consider the dynamical system in discrete time:

$$\begin{cases} x(t+dt) = x(t) + x'(t)dt \\ x'(t+dt) = x'(t) + x''(t)dt. \end{cases} \tag{25}$$

5 User Guide for the java executable file

If you use the **SVMVC.jar** file, you can only use the software GUI mode.

The software is formed by two main windows: the console window and the display window. The **console window** allows the modification of the parameters, start and stop approximation, save and load the results, control the system. The **display window** allows one to visualize the approximation of the viability kernel or the resilience values, and trajectories if the user chooses to control the system.

The next subsection describes the different parts of the console window, and all the parameters that can be chosen by the user.

5.1 Console window

The console window contains 7 parts. When modifying one parameter, you have to go inside a text box of the second part (that defines the parameters of the model) and press enter, in order to take into account the modifications.

5.1.1 Choosing the system

The first part, at the top left, allows to choose the dynamical system.

Figure 2: First part of the console window

5.1.2 Definition of parameters of the models

The second part, called *Population* in the figure 1, allows the definition of the viability constraint set and the parameters of the model.

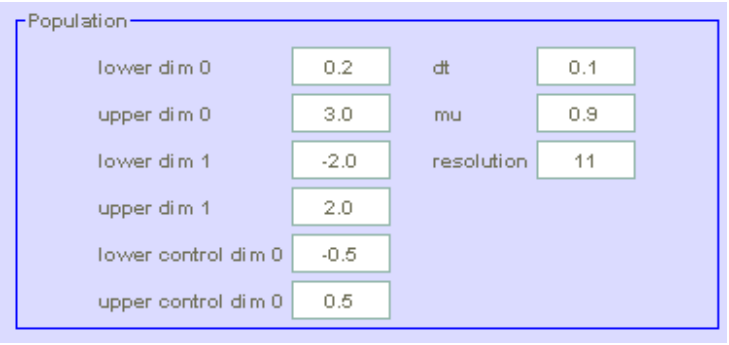

Figure 3: Second part of the console window

Table 1 describes the different parts of the window. There are as many text boxes as needed to choose the lower and upper bound of K and the control space. If the model has parameters, there are also special labels and text boxes to choose their values (not represented in the figure 3).

Element	Label	Description
text box	$lower\ dim\ i$	lower bound of K for the dimension i
text box	$upper\ dim\ i$	upper bound of K for the dimension i
text box	$lower\ control\ dim\ j$	lower bound of the j^{th} control dimension
text box	$upper\ control\ dim\ j$	upper bound of the j^{th} control dimension
label	$label\ of\ parameter\ k$	label of the k^{th} parameter
text box	$value\ of\ parameter\ k$	value of the k^{th} parameter
text box	dt	simulation time step
text box	mu	parameter for the gradient descent (a small value makes the simulation longer, a large value makes the simulation shorter but the optimization can be false)
text box	$resolution$	number of points by dimension

Table 1: Details of the second part of the console window

5.1.3 Viability controller

The third part, called the *Viability controller*, allows one to choose the number of steps (instead of looking if the system is viable at the next time step, we can look if it is viable at n time steps), and the number of steps at the first iteration (at the first iteration, the procedure is less robust, and it is better to use few time steps). It also allows the visualization the trajectories starting from each point of the grid, by checking *trajectories?*.

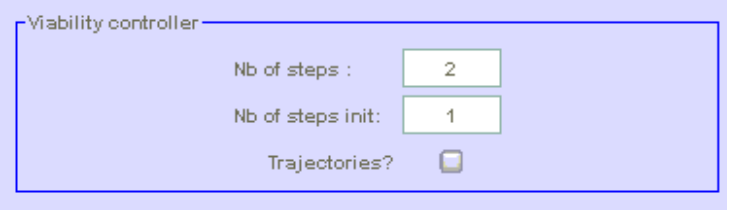

Figure 4: Third part of the console window

5.1.4 SVM configuration

The fourth part concerns the configuration of the SVM. To compute the SVM function, we use the library LibSvm (for more details about the library and its parameter settings, go to `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`).

You can choose some default parameters, detailed in figure 5. Parameter *epsilon* sets the tolerance of the termination criterion for the SVM computation algorithm. Its value must be small. The *SVM type* to choose is C. The value of C must be large, in order to fulfill the condition of the theorem (for more information about the theorem, see [Deffuant et al., 2007]). With the *gaussian kernel*, the SVM function can approximate all the classification functions (only use this kernel, the others are not functional). With the gaussian kernel, you have to choose the *gamma* (γ, see glossary) parameter, that

allows the control of the shape of the SVM function: a small value of *gamma* gives smooth shapes while a large value allows the approximation of more irregular shapes.

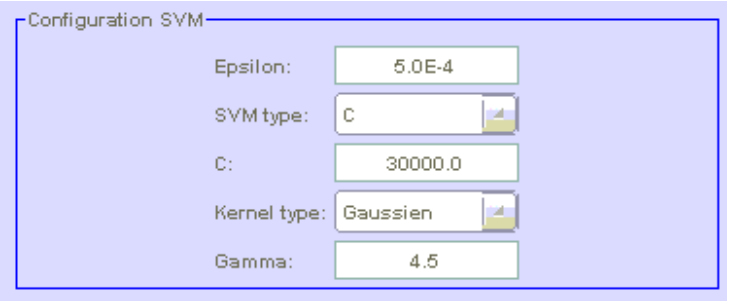

Figure 5: Fourth part of the console window

5.1.5 Execution and control

The fifth part concerns the execution.

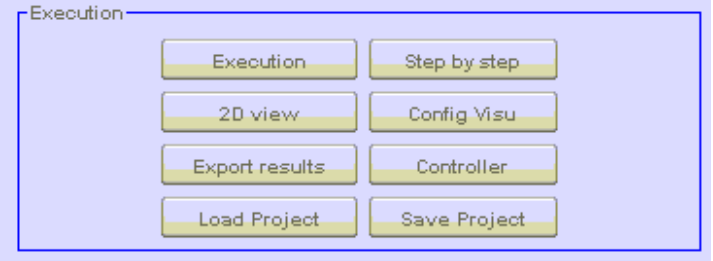

Figure 6: Fifth part of the console window

Table 2 describes the different buttons of the fifth part.

Button	**Description**
Execution	runs the program until the final approximation of the viability kernel is reached or the resilience values are computed
Step by step	shows all the iterations of the algorithm
2D view	activates or deactivates the display window
Config Visu	configures the display, by opening a configuration display window (see next paragraph (display configuration window))
Export results	export the results in a *.txt* file (see next paragraph (Export file structure))
Controller	configures the controller, by opening a controller configuration window (see next paragraph (controller configuration window))
Load project	loads an approximation you have already made and saved
Save project	saves the results in 2 files: *.log* and *.svm* files.

Table 2: Details of the second part of the console window

Display configuration window

Figure 7 presents the window of the configuration display. In this window, you can choose the dimension to display, and the value of the other dimensions for the display of the slices in 2D. If you modify the values or the dimensions to display, you have to press the *Update 2D view* to see the results on the display window. You can also activate or deactivate the view of the *theoretical curves* of the viability kernel (if they are available). You can choose whether or not to display the points of the grid. You can also only visualize the points that are support vectors by pressing the *Only SV* button. For the resilience values computation, you'll have another window (on the right of figure 7). Here, you can choose the number of level lines of the cost you want to display and the difference between two level lines.

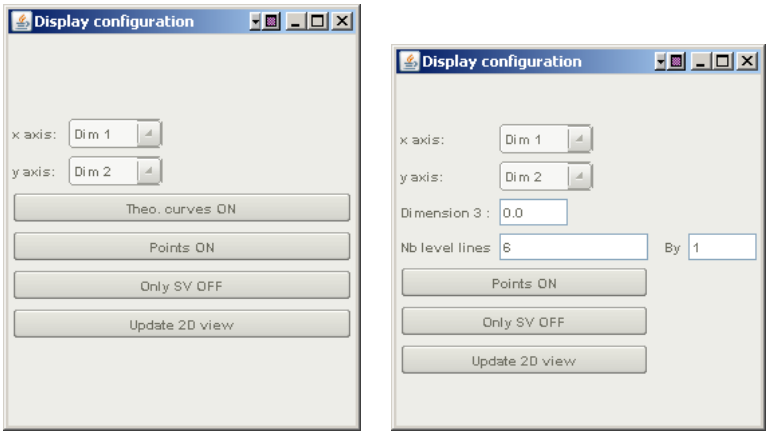

Figure 7: Configuration display window

Export file structure

The $.txt$ file contains the list of grid points and the SVM function parameters. The first part of the file is devoted to the list of the grid points. Each line corresponds to a point: each coordinate for all the dimensions are separated by a tab, and at the end of each line, separated by a tab from the coordinate of the last dimension, the label of the point ($+1$ if the point is viable, -1 if it is not). The list of points and the SVM function parameters are separated by an empty line. The SVM function is defined by a list of support vectors points: one point by line, coordinates separated by tab, at the end of each line, separated by a tab from the coordinates, the value of α_i (see glossary). The last line of the line corresponds to the value of b (see glossary).

Controller configuration window

Figure 8 presents the window of the configuration display.

This window allows the definition of the parameters of the controller (heavy controller here). To use this button, you must have already run the algorithm until the final approximation was reached. In this window, you choose the starting point of the trajectory (scaled in the space $[0, 1]^d$). You also choose the number of time steps of the trajectory. The check box *cautious policy* indicates that this is a heavy controller and that you have to choose the *number of time steps of anticipation* and the value of the SVM to define the *security distance*. This value must be greater than -1. A value of -1 value indicates that you allow no security distance (it correspond to the SVM value

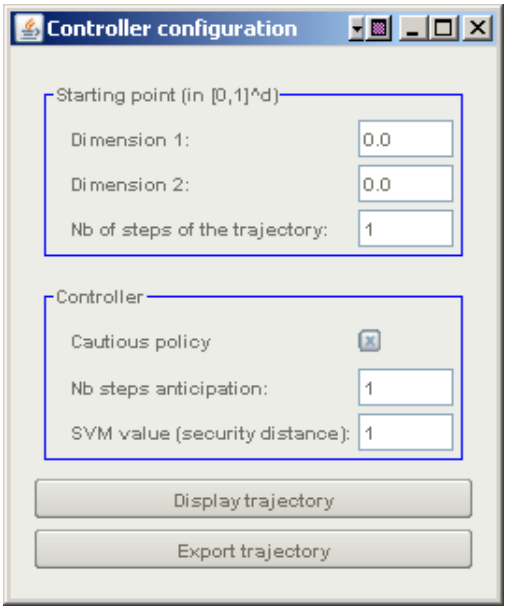

Figure 8: Viability controller window

that define the boundary of the approximation). The value to choose depends on the security distance you want, and also on the number of points in each dimension. The finer the grid, the larger the SVM value must be. The greater the security distance, the bigger the value of the SVM must be. You can check the distance obtained in the display window, the security distance is colored light blue. Press the *Display trajectory* button to visualize the result. Press the *Export trajectory* button to export the list of points of the trajectory in a *.txt* file. This file contains the list of points, one line by point, and the coordinates are separated by tab.

5.1.6 Indicators

The sixth part gives some indicators: the current *number of iterations* (for a step by step execution) or the final number of iterations (for execution mode), the *number of learns* (correspond to the number of computation of the SVM function, it is the same that the number of iterations in the version 1.0 of the algorithm, but will be different when the later version, including active learning, would be available), the final *time* of the approximation, the *number of points* of the whole grid, the *number of support vectors* of the current SVM (step by step mode) or the number of SVs of the final SVM (execution mode). The variable *M1* gives an indication of the progress of the algorithm. It indicates the number of grid points that have had their labels changed, relative to the size of the whole grid. A small value can mean that this is one of the last iterations of the approximation.

5.1.7 Log of the execution

The seventh part is the log of the execution. It summarizes the parameters chosen for the approximation, and during the run, indicates the progression of the algorithm. When

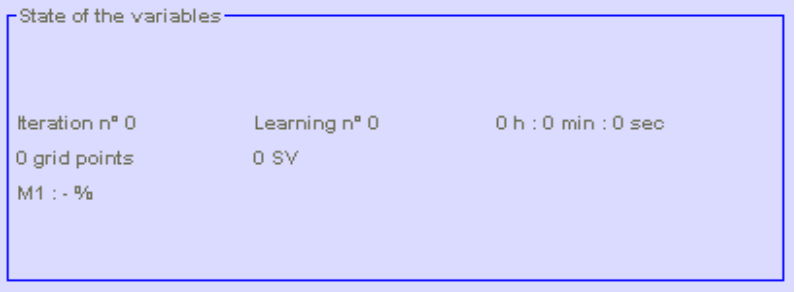

Figure 9: Sixth part of the console window

the system is being controlled, it also indicates the current point of the trajectory and the control used.

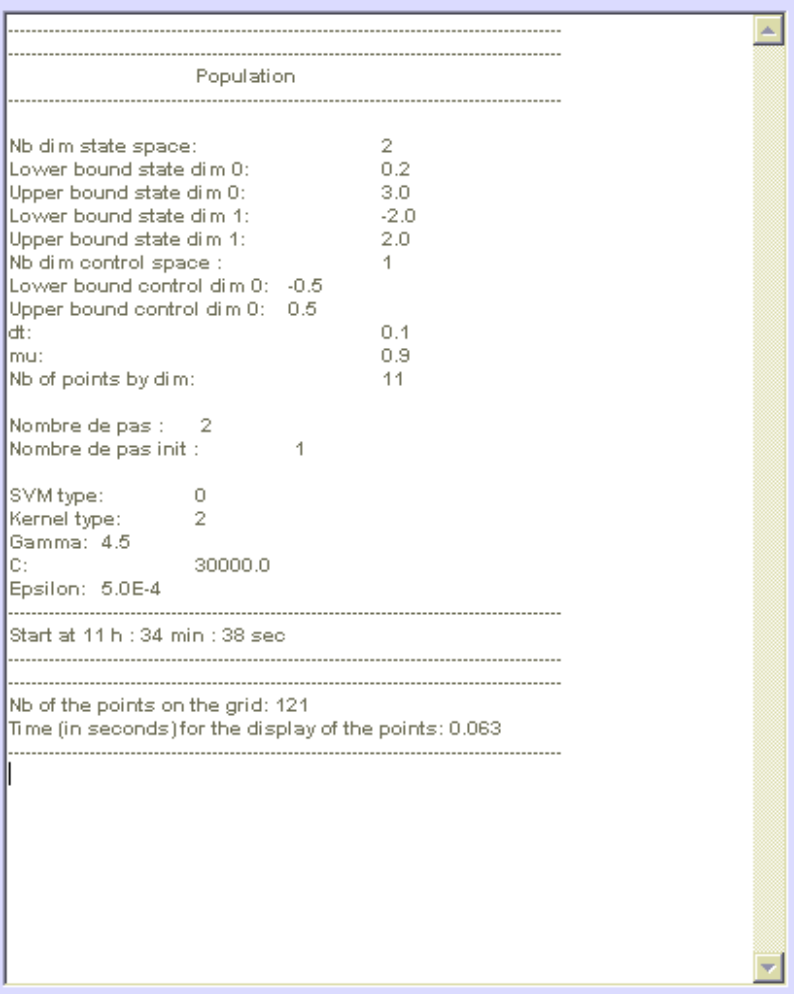

Figure 10: Seventh part of the console window

5.2 Display window

The display window represents, in 2D, the current or the final approximation of the viability kernel. The viability kernel is represented in dark blue. The points of the grid are also displayed: a yellow point represents a viable point and a red point, a non-viable point. The black lines represent the boundary of the real viability kernel (if the theoretical curves are available). If resilience is computated and if chosen on the configuration display window, the level lines of the resilience values are drawn. The colors of the level lines are a function of the resilience values: blue (high resilience value) to red (low resilience value). The given resilience values does not correspond to the value of the point x, but to the value of an initial point that will jump to that point x.

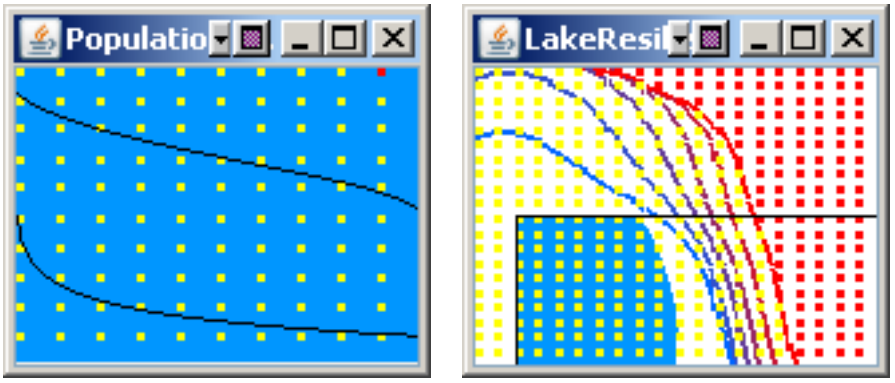

Figure 11: Display window

To change the size of the window, drag the window corner to enlarge or diminish it and click 3 times on the drawing. To zoom without changing the size of the window, click 2 times. To come back to the initial size, click 3 times.

6 User guide for adding a dynamical system

All the implementations must be only made in the package *Model*.
First, in the class *Model.java*:

- in the array *names_*, add the name of your new dynamical model;
- in the method *GetSytem(String inName)*, add a test for your new model.

After that, you have to create a new class file, with the name of your model (for example, *MyClass.java*), in the package *Model*. Depending on the results you want, you have to extend your class from:

- *Dynamic_System* if you want to approximate viability kernels;
- *Dynamic_System_Target* if you want to approximate capture basins;
- *Dynamic_System_Resilience* if you want to compute resilience values.

6.1 Approximating viability kernels

You have to define the constructor of *MyClass*, with the following fields in the same order:

N°	Fields	Type	Comments
1	$name\_$	String	name of the system
2	$nbDim\_$	int	dimension of the state space
	$nbDimControl\_$	int	dimension of the control space
	$nbDimParam\_$	int	number of parameters
3	$initSystem()$	method	allow to initialize arrays of right size
4	$lowerBoundState\_[i]$	float	lower bound of the i^{th} dimension of K
	$upperBoundState\_[i]$	float	upper bound of the i^{th} dimension of K
	$lowerBoundControl\_[j]$	float	lower bound of the j^{th} control
	$upperBoundControl\_[j]$	float	upper bound of the j^{th} control
	$labelParam\_[k]$	String	label of the k^{th} parameter
	$valueParam\_[k]$	float	value of the k^{th} parameter
5	$mu\_$	float	param of the gradient descent
	$resolution\_$	int	number of points by dimension
	$dt\_$	float	dt value
	$theoricCurves\_$	boolean	true if theoretical curves available

Table 3: Fields for *Dynamic_System*

The type *MyClass.java* must implement the following inherited abstract method:

Method	Params	Returns
computePhi(Point inP, Point inU, double dt) *compute* $\varphi(x(t), u(t))$	inP: $x(t)$ inU: $u(t)$ dt: dt value	Point $\varphi(x(t), u(t))$

6.2 Approximating capture basins

You have to define the constructor of *MyClass*, with the fields defined in table 3.

The type *MyClass.java* must implement the following inherited abstract method:

Method	**Params**	**Returns**
computePhi(Point inP, Point inU, double dt) *compute* $\varphi(x(t), u(t))$	inP: $x(t)$ inU: $u(t)$ dt: dt value	Point $\varphi(x(t), u(t))$
centerOfTheTarget(int i) *gives the* i^{th} *coordinate of the center of the target*	i: dimension	float i^{th} coordinate
isInTheTarget(Point inP) *tests if* $x(t) \in C$	inP: $x(t)$	boolean true if $x(t) \in C$

6.3 Computing resilience values

You have to define the constructor of *MyClass*, with the fields defined in table 3 plus (in the right order):

N°	**Fields**	**Type**	**Comments**
2	$nbCost\_$	int	number of parameters for the cost
4	$lowerK\_[l]$	float	lower bound of the l^{th} dimension of K
	$upperK\_[l]$	float	upper bound of the l^{th} dimension of K with $l \leq (nbDim\_ - 1)$
	$valueCost\_[m]$	float	value of the m^{th} cost

Table 4: tab:Fields for *Dynamic_System_Resilience*

The type *MyClass.java* must implement the following inherited abstract method:

Method	**Params**	**Returns**
computePhi(Point inP, Point inU, double dt) *compute* $\varphi(x(t), u(t))$	inP: $x(t)$ inU: $u(t)$ dt: dt value	Point $\varphi(x(t), u(t))$
computeCost(Point inP) *gives the cost of a point* $x(t)$	inP: $x(t)$	float cost for $x(t)$

7 User guide for running the program in the batch mode

7.1 Running the program

To run the software in a batch mode, files with extension $.simu$ are needed. They contain all the information needed. The next subsection details the construction of a $.simu$ file.

To run the program in a batch mode, write the following instruction in a command window:

```
C:\SVMVC v1.0\bin > java Appli/Batch Conso.simu
```

Replace $Conso.simu$ by the name of your file.
At the end of the execution, you'll have the results written in the command window:

```
Execution --> Conso Consumption December 11, 2007 5-31-44 PM CET/
Results
9 iterations
9 learns
18 SV
961 points
Computed in 0 h 0 m 9 s.
```

The execution creates 2 files: one with a $.svm$ extension, one with a $.log$ extension. The text in the console indicates which directory the files are saved in (here, in a directory called *Conso Consumption December 11, 2007 5-31-44 PM CET/*), the number of iterations and learns needed to obtain the final approximation, the number of support vectors of the last approximation and the size of the whole grid. It also indicates the time needed to compute the final approximation. To visualize the approximation, you have to run the program in the GUI mode, either by using the $.jar$ file or by executing the following instruction in a command window:

```
C:\SVMVC v1.0\bin > java Appli/GUI
```

In the GUI console, use the *load project* button and select the $.svm$ file to visualize the results.

The batch mode also allows one to get the details of all the iterations, by using the parameter $-v$ in the instructions (equivalent to the step by step execution in GUI mode):

```
C:\SVMVC v1.0\bin > java Appli/Batch Conso.simu -v
```

At the end of the execution, you'll have the results written:

```
Execution --> Conso Consumption December 11, 2007 5-36-28 PM CET/
Iter. 1 - M1 = 23.51717%
Iter. 2 - M1 = 9.781478%
Iter. 3 - M1 = 3.850156%
Iter. 4 - M1 = 1.2486992%
Iter. 5 - M1 = 0.7284079%
Iter. 6 - M1 = 0.41623312%
Iter. 7 - M1 = 0.20811656%
Iter. 8 - M1 = 0.10405828%
Iter. 9 - M1 = 0.0%
Results
```

```
9 iterations
9 learns
18 SV
961 points
Computed in 0 h 0 m 10 s.
```

Here, there are as many $.simu$ and $.svm$ files saved as the number of iterations.

7.2 Constructing a .simu file

The $.simu$ files must be carefully constructed. These files are composed of lines (one line per parameter to define). For each line, indicate the name of the parameter you want to define, and their values must be located at the right of ":". The names of the parameters are not important (they are only for recall), but they have to be defined in the right order. Table 5 gives the different components of the file.

Lines	**Description**
$name$	name of the system, such that written in the attribute $names\_$ of the $Model$ class
$lower\ K\ i$ $upper\ K\ i$	value of the lower bound of K for dimension i (only for resilience computation) value of the upper bound of K for dimension i (only for resilience computation)
Comments	first lower and upper for dimension 1, lower and upper for dimension 2 etc.
$cost\ j$	value of the j^{th} cost (only for resilience computation)
$lower\ dim\ k$ $upper\ dim\ k$	lower bound of K for the dimension k (lower bound of H if resilience computation) upper bound of K for the dimension k (upper bound of H if resilience computation)
Comments	first lower and upper for dimension 1, lower and upper for dimension 2 etc.
$lower\ control\ dim\ l$ $upper\ control\ dim\ l$	lower bound of the l^{th} control dimension upper bound of the l^{th} control dimension
$value\ of\ parameter\ m$	value of the m^{th} parameter
dt mu $resolution$	simulation time step parameter for the gradient descent number of points by dimension for the simulation
$epsilon$ $svm\ type$ C $kernel\ type$ $gamma$	tolerance of termination criterion for SVM computation 0 for a C-SCM value of C parameter, if C-SVM 2 for a gaussian kernel gamma parameter for the SVM
$number\ of\ steps$ $number\ of\ steps\ init$	number of steps number of steps at the first iteration

Table 5: Details of the parameters of a $.simu$ file

The following code gives an example of a $.simu$ file for the lakeResilience system:

```
Name            : LakeResilience
lower K 0 : 0.1
upper K 0 : 1
lower K 1 : 0
upper K 1 : 0.5
C1 : 1.5
C2 : 5
lower dim 0 : 0
upper dim 0 : 1
lower dim 1 : 0
upper dim 1 : 1
lower dim 2 : 0
upper dim 2 : 6
lower control 0 : -1
upper control 0 : 1
q : 8
r : 1
m : 1
b : 0.8
dt : 0.1
mu : 0.9
resolution : 11
epsilon        : 0.0005
svm type      : 0
c              : 30000
kernel type     : 2
gamma         : 4
nb steps         : 2
nb steps init    : 1
```

References

[Abrams and Strogatz, 2003] Abrams, D. and Strogatz, S. (2003). Modelling the dynamics of language death. *Nature*, 424(6951):900–900.

[Aubin, 1991] Aubin, J.-P. (1991). *Viability theory*. Birkhäuser.

[Aubin, 2002] Aubin, J.-P. (2002). An introduction to viability theory and management of renewable resources. In *Coupling Climate and Economic Dynamics, Geneva*.

[Bernard et al., 2007] Bernard, C., Chapel, L., Deffuant, G., Martin, S., and San-Miguel, M. (2007). Language resilience in Abrams-Strogatz and Minett-Wang models. Working paper - european project PATRES.

[Cardaliaguet et al., 1997] Cardaliaguet, P., Quincampoix, M., and Saint-Pierre, P. (1997). Optimal times for constrained nonlinear control problems without local optimality. *Applied Mathematics & Optimization*, 36:21–42.

[Carpenter et al., 1999] Carpenter, S.-R., Brock, W., and Hanson, P. (1999). Ecological and social dynamics in simple models of ecosystem management. *Conservation Ecology*, 3.

[Deffuant et al., 2007] Deffuant, G., Chapel, L., and Martin, S. (2007). Approximating viability kernels with support vector machines. *IEEE transactions on automatic control*, 52(5):933–937.

[Martin, 2004] Martin, S. (2004). The cost of restoration as a way of defining resilience: a viability approach applied to a model of lake eutrophication. *Ecology and Society*, 9(2).

[Minett and Wang, IP] Minett, J. W. and Wang, W. S.-Y. (IP). Modelling endangered languages: The effects of bilingualism and social structure. *Lingua*.

[Moore and Atkeson, 1995] Moore, A. and Atkeson, C. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233.