ENSTA Bretagne
2, rue François Verny
29806 BREST cedex
FRANCE
Tel +33 (0)2 98 34 88 00
www.ensta-bretagne.fr

Rapport de stage Assistant Ingénieur

Stability Analysis using Interval Methods

University Of Manchester

Walter Jean
Option SPID – Profil Robotique
Jean.Walter@ensta-bretagne.org

Résumé – Abstract

Ce rapport propose de quantifier à quel point la régulation par mode glissant d'un robot est robuste lorsque ses équations d'états contiennent des incertitudes. Afin de quantifier cette robustesse, une analyse de stabilité du système sera effectuée, notamment en adoptant une approche de V-stabilité. Lors de cette analyse de stabilité, un problème de borne de fonction se posera. Ce rapport présente donc également comment borner des fonctions dépendant de plusieurs paramètres (6 en l'occurrence) à l'aide d'une méthode d'analyse d'intervalle. Un SIVIA permettra ensuite de vérifier dans quelles circonstances les conditions de V-stabilité sont remplies, ce qui permettra d'identifier quelles valeurs de paramètre sont judicieuses pour dimensionner le régulateur. Plusieurs surfaces de régulation par mode glissant seront testées afin d'évaluer également l'influence de ces dernières sur la stabilité du système. Une régulation par bouclage linéarisant sera également étudiée afin de comparer les deux modes de régulation et de déterminer si l'un des deux est plus robuste que l'autre, et si oui, dans quelles conditions.

This report proposes to quantify the robustness of sliding mode control when the states equations of the robot are subject to uncertainties. In order to quantify this robustness, a stability analysis of the system will be performed by adopting a V-stability approach. By doing a stability analysis, at some point a function depending of several parameters will have to be bound. Thus, the report also presents how to bound a function with an interval analysis approach. A SIVIA method will enabled to verify in which cases the conditions of V-stability are consistent, it will give the values of parameters which are needed to dimension the controller. Several sliding mode surfaces will be tested to evaluate the influence the influence of them on the stability of the system. A feedback linearization will then be performed to compare which regulation is the most robust and under which conditions.

Table of content

1 - Introduction

As a student from ENSTA Bretagne getting a training about robotic systems, I get the opportunity to do a research internship in the University of Manchester during the summer 2017 for twelve weeks. The University of Manchester groups many fields, from art study to aerospace engineering, and carry research in sectors like material, energy, autonomous systems etc… This internship was for me the occasion to apply and to develop my skill in the robotic field, especially in interval analysis, and to discover what is it like to work in the research sector. This report aims to describe the conditions in which I was during this internship, what I have done and to give an analysis of my work. Firstly, I will describe the University of Manchester and the department I worked in, then explain the project I worked on, detail what I have done and my results before concluding with what this internship brought me.

2 -The University of Manchester

The University of Manchester was created in 1824 and currently represents the largest student community in the United Kingdom with almost 40 000 students, 30 000 undergraduates and 12 000 postgraduates. The university is currently 54th in the World University Rankings 2018 and counts 25 Nobel Prize winners.

Concerning my internship, I was in the department "School of Electrical and Electronic Engineering" created in 1905. This department groups several research areas such as control theory, autonomous systems, robotics for the nuclear industry, process control etc… I worked in the autonomous systems research area under the directives of Dr Alexandru Stancu. The aim of this department is to develop technologies that require minimal human intervention to be operational. The team was composed of Dr Stancu two other doctors and three PhD students. Concerning the organisation of the team, Dr Stancu was the leader. He gave directives to PhD students. Those PhD students, in addition to work on their thesis, helped MSc students in their summer project. During the internship, I worked on stability analysis using interval method. In order to follow the advance of my work during my traineeship, we organised with Dr Stancu and PhD student meetings in his office once a week at least to discuss about my work and results. Whenever I had questions about something, I could ask the PhD students who were working next to me in an open space.

3 - Problem statement

Consider a robot described by a state equation

$$\dot{x} = f(x, p)$$

Where $x \in \mathbb{R}^n$ is the state vector, $p \in \mathbb{R}^m$ is a vector containing the uncertainties of the system, and f the evolution function. In order to assure that the robot will be operational, a stability analysis has to be done to define in which condition the system is no longer safe. Quantifying the "stable area" i.e. an area where the different parameters verify the V-stability approach could be interesting. Indeed, if it is possible to display the stable area, the safety of the system could be guaranteed despite uncertainties. It also gives information, for the user, about how to dimension the controller. For example, consider a sliding mode control applied with the following dynamic for the error:

$$\dot{s} = -Qs - Psign(s) + \psi$$

P and Q affect either the behaviour of the robot, and its stability and ψ is a function resulting of uncertainties in the state equation of the robot. The objective of the project would be to determine the safe area, depending of the vector s, and to evaluate the impact of P and Q on the stability. Thus, optimal values for these variables could be find and the operationality of the robot could be assure for some values of its state equation's parameters. Then these values could be used by the operator who knows in which condition the system is stable or not.

To resume, the aim of the project is to demonstrate the robustness of the sliding mode control by adding uncertainties in the states equations and to plot the stable area. This stable area verifies the V-stability approach. In order to solve non-linear problem and to plot this stable area, an interval analysis approach can be used.

4 - Sliding mode control with uncertainties in the state equations

4.1 Sliding mode without uncertainties

Firstly, consider a sliding mode without uncertainty. Consider two robots [1]. The first is described by

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} u_1 \cos\theta \\ u_1 \sin\theta \\ u_2 \end{pmatrix}$$

The second one

$$\begin{pmatrix} \dot{x}_d \\ \dot{y}_d \\ \dot{\theta}_d \end{pmatrix} = \begin{pmatrix} u_{d1} \cos\theta_d \\ u_{d1} \sin\theta_d \\ u_{d2} \end{pmatrix}$$

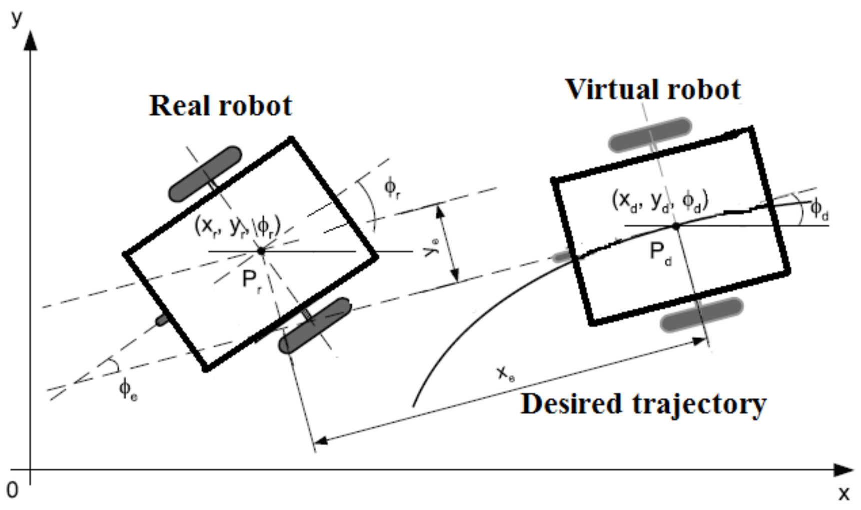


Figure 1 : Representation of the two robots

The tracking error is defined by

$$\begin{pmatrix} x_e \\ y_e \\ \theta_e \end{pmatrix} = \begin{pmatrix} \cos\theta_d & \sin\theta_d & 0 \\ -\sin\theta_d & \cos\theta_d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x - x_d \\ y - y_d \\ \theta - \theta_d \end{pmatrix}.$$

Thus,

$$\begin{pmatrix} x - x_d \\ y - y_d \\ \theta - \theta_d \end{pmatrix} = \begin{pmatrix} \cos\theta_d & -\sin\theta_d & 0 \\ \sin\theta_d & \cos\theta_d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ \theta_e \end{pmatrix}$$

After derivation with respect to t, the tracking error becomes:

$$\begin{aligned}
\begin{pmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{pmatrix} &= \begin{pmatrix} \cos\theta_d & \sin\theta_d & 0 \\ -\sin\theta_d & \cos\theta_d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \dot{x} - \dot{x}_d \\ \dot{y} - \dot{y}_d \\ \dot{\theta} - \dot{\theta}_d \end{pmatrix} + \begin{pmatrix} -u_{2d}\sin\theta_d & u_{2d}\cos\theta_d & 0 \\ -u_{2d}\cos\theta_d & -u_{2d}\sin\theta_d & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x - x_d \\ y - y_d \\ \theta - \theta_d \end{pmatrix} \\
&= \begin{pmatrix} \cos\theta_d & \sin\theta_d & 0 \\ -\sin\theta_d & \cos\theta_d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1\cos\theta - u_{d1}\cos\theta_d \\ u_1\sin\theta - u_{d1}\sin\theta_d \\ u_2 - u_{d2} \end{pmatrix} + \begin{pmatrix} -u_{2d}\sin\theta_d & u_{2d}\cos\theta_d & 0 \\ -u_{2d}\cos\theta_d & -u_{2d}\sin\theta_d & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x - x_d \\ y - y_d \\ \theta - \theta_d \end{pmatrix} \\
&= \underbrace{\begin{pmatrix} \cos\theta_d & \sin\theta_d & 0 \\ -\sin\theta_d & \cos\theta_d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1\cos\theta - u_{d1}\cos\theta_d \\ u_1\sin\theta - u_{d1}\sin\theta_d \\ u_2 - u_{d2} \end{pmatrix}}_{= \begin{pmatrix} -u_{1d} + u_1\cos(\theta - \theta_d) \\ u_1\sin(\theta - \theta_d) \\ u_2 - u_{2d} \end{pmatrix}} \\
&\quad + \underbrace{\begin{pmatrix} -u_{2d}\sin\theta_d & u_{2d}\cos\theta_d & 0 \\ -u_{2d}\cos\theta_d & -u_{2d}\sin\theta_d & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \cos\theta_d & -\sin\theta_d & 0 \\ \sin\theta_d & \cos\theta_d & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\begin{pmatrix} 0 & u_{2d} & 0 \\ -u_{2d} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}} \begin{pmatrix} x_e \\ y_e \\ \theta_e \end{pmatrix} \\
&= \begin{pmatrix} -u_{1d} + u_1\cos(\theta_e) \\ u_1\sin\theta_e \\ u_2 - u_{2d} \end{pmatrix} + \begin{pmatrix} u_{2d}y_e \\ -u_{2d}x_e \\ 0 \end{pmatrix}
\end{aligned}$$

Finally,

$$\begin{pmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{pmatrix} = \begin{pmatrix} -u_{1d} + u_1\cos\theta_e + u_{2d}y_e \\ u_1\sin\theta_e - u_{2d}x_e \\ u_2 - u_{2d} \end{pmatrix}$$

The following s_1 and s_2 equations are the two surfaces in the error space

$$\begin{aligned} s_1 &= \dot{x}_e + x_e \\ s_2 &= \dot{y}_e + y_e \end{aligned}$$

Which is equivalent to:

$$\mathbf{s} = \begin{pmatrix} -u_{1d} + u_1\cos\theta_e + u_{2d}y_e + x_e \\ u_1\sin\theta_e - u_{2d}x_e + y_e \end{pmatrix}$$

Thus,

$$\begin{aligned}
\dot{\mathbf{s}} &= \begin{pmatrix} -\dot{u}_{1d} + \dot{u}_1\cos\theta_e - u_1\dot{\theta}_e\sin\theta_e + \dot{u}_{2d}y_e + u_{2d}\dot{y}_e + \dot{x}_e \\ \dot{u}_1\sin\theta_e + u_1\dot{\theta}_e\cos\theta_e - \dot{u}_{2d}x_e - u_{2d}\dot{x}_e + \dot{y}_e \end{pmatrix} \\
&= \begin{pmatrix} -\dot{u}_{1d} + \dot{u}_1\cos\theta_e - u_1(u_2 - u_{2d})\sin\theta_e + \dot{u}_{2d}y_e + u_{2d}(u_1\sin\theta_e - u_{2d}x_e) - u_{1d} + u_1\cos\theta_e + u_{2d}y_e \\ \dot{u}_1\sin\theta_e + u_1(u_2 - u_{2d})\cos\theta_e - \dot{u}_{2d}x_e - u_{2d}(-u_{1d} + u_1\cos\theta_e + u_{2d}y_e) + u_1\sin\theta_e - u_{2d}x_e \end{pmatrix}
\end{aligned}$$

$$\dot{\mathbf{s}} = \underbrace{\begin{pmatrix} \cos\theta_e & -u_1 \sin\theta_e \\ \sin\theta_e & u_1\cos\theta_e \end{pmatrix}}_{\mathbf{A}(u_1,,\theta_e,\mathbf{u}_d)} \begin{pmatrix} \dot{u}_1 \\ u_2 \end{pmatrix} + \underbrace{\begin{pmatrix} -\dot{u}_{1d} + u_1u_{2d}\sin\theta_e + \dot{u}_{2d}y_e + u_{2d}\left(u_1\sin\theta_e - u_{2d}x_e\right) - u_{1d} + u_1\cos\theta_e + u_{2d}y_e \\ -u_1u_{2d}\cos\theta_e - \dot{u}_{2d}x_e - u_{2d}\left(-u_{1d} + u_1\cos\theta_e + u_{2d}y_e\right) + u_1\sin\theta_e - u_{2d}x_e \end{pmatrix}}_{\mathbf{b}(u_1,x_e,y_e,\theta_e,\mathbf{u}_d)}$$

i.e.

$$\dot{\mathbf{s}} = \mathbf{A}\left(u_1, \theta_e, \mathbf{u}_d\right) . \begin{pmatrix} \dot{u}_1 \\ u_2 \end{pmatrix} + \mathbf{b}\left(u_1, x_e, y_e, \theta_e, \mathbf{u}_d\right) .$$

Consider the following dynamic for the error:

$$\dot{s} = -Qs - Psign(s)$$

Then,

$$\mathbf{A}\left(u_1, \theta_e, \mathbf{u}_d\right) . \begin{pmatrix} \dot{u}_1 \\ u_2 \end{pmatrix} + \mathbf{b}\left(u_1, x_e, y_e, \theta_e, \mathbf{u}_d\right) \quad = -Qs - Psign(s)$$

i.e.

$$\begin{pmatrix} \dot{u}_1 \\ u_2 \end{pmatrix} = \mathbf{A}^{-1}\left(u_1, \theta_e, \mathbf{u}_d\right) \left(-Qs - Psign(s) \quad - \mathbf{b}\left(u_1, x_e, y_e, \theta_e, \mathbf{u}_d\right)\right) .$$

In the equation $\dot{s} = -Qs - Psign(s)$, Q affects the way the system reaches the sliding mode surface. The bigger Q is, the fastest the error will be equal to zero. P has an impact on the shattering effect. Indeed, with the sliding mode control, whenever the system is out of the sliding mode surface, the $Psign(s)$ term corrects the trajectory into the surface. The bigger P is, the stronger the correction will be. However, too much shattering could lead to the instability of the robot. Effectively, if the correction is too strong, the regulation could make the robot pass the desired trajectory in such a way it will not be able to reach the desired trajectory again.

4.2 V-Stability applied to sliding mode with uncertainties

4.2.1 V-stability of a system

Definition of V-Stability:

Consider a robot defined by the following state equation:

$$\dot{x} = f(x)$$

Consider a differential inclusion V, by definition a system is V-stable if:

$$V(x) \geq 0 \implies \dot{V}(x) \leq -\varepsilon$$

With $\varepsilon > 0$ [2].

Application to the sliding mode control:

Assume now that the actual robot has some very small uncertainty. For instance

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} (u1 + p1)\cos(\theta) \\ (u1 + p1)\sin(\theta) \\ u2 + p2 \end{pmatrix}$$

With $p_i \,\epsilon\, [-0.1\,; 0.1]$.

By applying the controller given above. The error equation becomes

$$\dot{s} = -Qs - Psign(s) + \psi(x_e, y_e, \theta_e, p)$$

To study the V-stability of this system, consider the following differential inclusion:

$$V(s) = s^T s - r$$

With $r \in \mathbb{R}^+$
After derivation with respect to s:

$$\dot{V} = 2(s_1\dot{s}_1 + s_2\dot{s}_2)$$

To be V-stable the following condition needs to be verified,

$$V(s) \geq 0 \implies \dot{V}(s) \leq -\varepsilon$$

The function $\dot{V}(s)$ can be bounded as below:

$$\dot{V} \in 2(s_1(-Qs_1 - Psign(s_1) + [\psi_1]([x_e],[y_e],[\theta_e],[p])\,) + s_2(-Qs_2 - Psign(s_2) + [\psi_2]([x_e],[y_e],[\theta_e],[p])\,))$$

To know more about the robustness of the sliding mode, ψ_1 and ψ_2 must be bounded. The two functions ψ_1 and ψ_2 are the differences between the sliding mode with uncertainties and without. With uncertainties, s becomes

$$s = \begin{pmatrix} -u_{1d} + (u_1 + p_1)\cos(\theta_e) + u_{2d}y_e + x_e \\ (u_1 + p_1)\sin(\theta_e) - u_{2d}x_e + y_e \end{pmatrix}$$

After doing the same computation as previously, $\dot{s}$ with uncertainties is equal to:

$$\dot{s} = \begin{pmatrix} -\dot{u_{1d}} + (\dot{u_1})\cos(\theta_e) - (u_1 + p_1)(u_2 + p_2 - u_{2d})\sin(\theta_e) + \dot{u_{2d}}y_e + u_{2d}\big((u_1 + p_1)\sin(\theta_e) - u_{2d}x_e\big) - u_{1d} + (u_1 + p_1)\cos(\theta_e) + u_{2d}y_e \\ \dot{u_1}\sin(\theta_e) + (u_1 + p_1)(u_2 + p_2 - u_{2d})\cos(\theta_e) - \dot{u_{2d}}x_e - u_{2d}(-u_{1d} + (u_1 + p_1)\cos(\theta_e) + u_{2d}y_e) + (u_1 + p_1)\sin(\theta_e) - u_{2d}x_e \end{pmatrix} \quad (1)$$

Recall: without uncertainties, $\dot{s}$ is equal to:

$$\dot{s} = \begin{pmatrix} -\dot{u}_{1d} + \dot{u}_1\cos\theta_e - u_1(u_2 - u_{2d})\sin\theta_e + \dot{u}_{2d}y_e + u_{2d}(u_1\sin\theta_e - u_{2d}x_e) - u_{1d} + u_1\cos\theta_e + u_{2d}y_e \\ \dot{u}_1\sin\theta_e + u_1(u_2 - u_{2d})\cos\theta_e - \dot{u}_{2d}x_e - u_{2d}(-u_{1d} + u_1\cos\theta_e + u_{2d}y_e) + u_1\sin\theta_e - u_{2d}x_e \end{pmatrix} \quad (2)$$

The two functions ψ_1 and ψ_2 are obtained after subtracting (2) to (1) :

$$\begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix} = \begin{pmatrix} \sin(\theta_e)(-p_1u_2 + 2p_1u_{2d} - p_2u_1 - p_1p_2) + p_1\cos(\theta_e) \\ \cos(\theta_e)(p_1u_2 - 2p_1u_{2d} + p_2u_1 + p_1p_2) + p_1\sin(\theta_e) \end{pmatrix}$$

Now that the expression of ψ_1 and ψ_2 are known, it is possible to determine their influence on the V-stability by bounding those functions.

4.2.2 Bounding a function using interval analysis

To bound ψ_1 and ψ_2, several methods can be used.

4.2.2.1 - First Method

The paragraph below describes a first method to find an upper bound.
To make the explanation of this method simpler, the function ψ will be in two-dimensions. The graph below represents ψ (in black) and the blue boxes are the estimation of the function in several intervals.

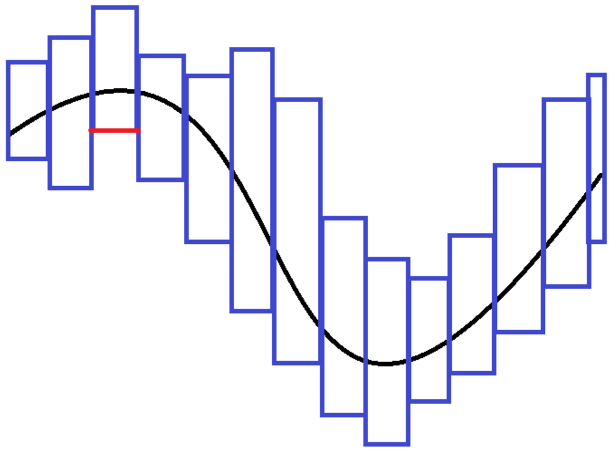


Figure 2: ψ and its estimation with boxes

Consider λ the maximum of the lower bound of all boxes (in red in the graph above). By definition, the upper bound is not in the boxes whose maximum is smaller than λ. Thus, those boxes can be eliminated and the remaining ones can be contracted until the width of the boxes is greater than ε. For instance, in the figure below, the clearer boxes will be suppressed.

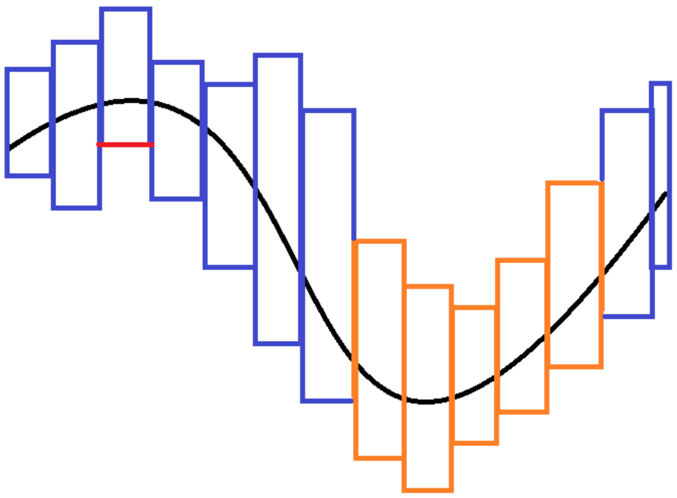


Figure 3 : the clearer boxes will be deleted

Consider now a function $\psi(x, p)$ with $x_i \in [-10\ ; 10]$ and $p_i \in [-10^{-5}; 10^{-5}]$. p_i is very small so it doesn't need to be bisected.

To find an upper bound the same approach can be used,

- Split the x_i in $k \epsilon \mathbb{N}$
- Estimate ψ on those intervals
- Compute λ
- Suppress the x_i box whose maximum is smaller than λ
- Split again the x_i while the width of the x_i box is greater than ε

The graph below is an example with $x = \binom{x_1}{x_2}$ split in 3.

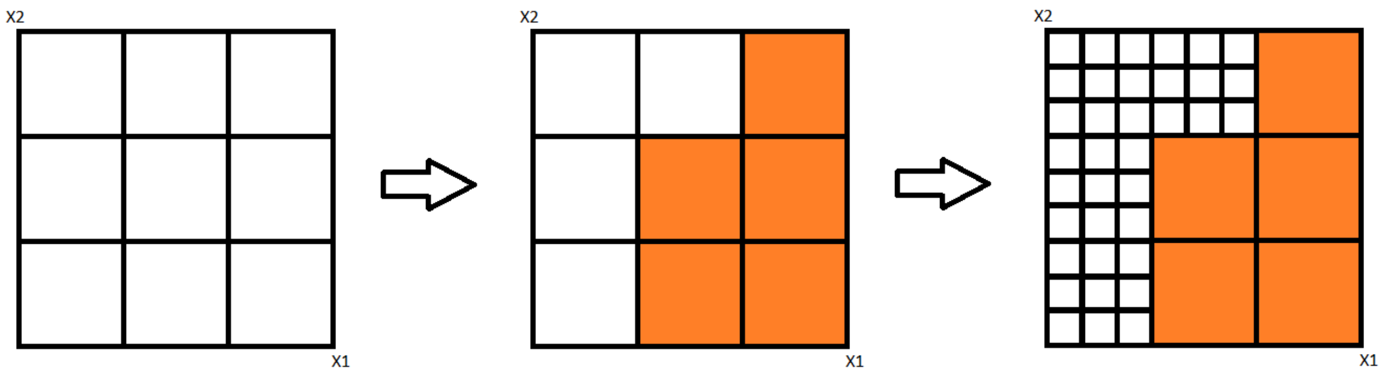


Figure 4: First method to find an upper bound and a lower bound by splitting by 3 x_1 and x_2

First, x_1 and x_2 are divided by $k = 3$. Then ψ is estimated for each box. After computing λ, the boxes which upper bound is smaller than λ (the clearer boxes from above for example) are suppressed. Then, the remaining boxes are split again until the width of the boxes is greater than ε.
However, after trying this method with the ψ_1 and ψ_2 equations, it appears that this method is not appropriate because there are too many parameters, it takes too much time to have a result.
Another method must be find.

4.2.2.2 Second method: MypySIVIA

4.2.2.2.1 Principle of the method MypySIVIA

Recall :

$$\begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} = \begin{pmatrix}\sin(\theta_e)\,(-p_1u_2 + 2p_1u_{2d} - p_2u_1 - p_1p_2) + p_1\cos(\theta_e)\\ \cos(\theta_e)\,(p_1u_2 - 2p_1u_{2d} + p_2u_1 + p_1p_2) + p_1\sin(\theta_e)\end{pmatrix}$$

Consider ψ_{max} the upper bound of ψ and $I(\psi_{max})$ the interval $]\,\psi_{max};\,+\infty\,[$. If SIVIA with a separator associated to the constraint $\psi \in I(\psi_{max})$ is applied, there won't be any solution because ψ_{max} is the upper bound. Thus, the objective of this method will be to find the first value, named f_{sup}, for which the separator associated to the constraint $\psi \in\ I\left(f_{sup}\right)$ will return no solution.

Replacing $\theta_e, u_2, p_1, u_{2d}, u_1, p_2$ with intervals gives a first overestimated approximation of the bound of ψ_1 and ψ_2. Consider *a* and *b* the lower and the upper bound of this first overestimation. Then consider a variable named *m* which is the bisection of the interval [*a ; b*]. If there is a solution after applying SIVIA with a separator associated to the constraint $\psi \in I(m)$, *m* is not an upper bound. In this case, the upper bound is greater than m, so *a* is replaced by *m*. Otherwise, if there is no solution, m is an upper bound but may be not the smallest upper bound, so *b* is replaced by *m*. Those operations are repeated until the width of [*a ; b*] is greater than ε.

The drawing below illustrates the behaviour of this method.

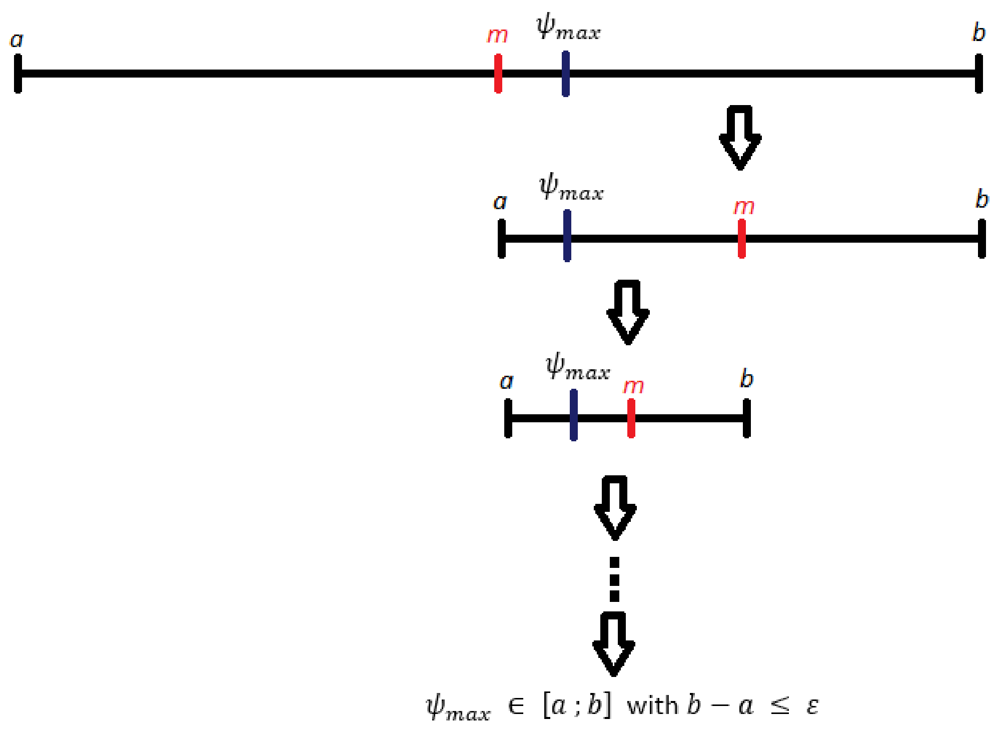


Figure 5: Illustration of how MypySIVIA method works

To obtain the lower bound, it is the same method but instead of using $I(\psi_{max}) \in]\,\psi_{max};\, +\infty\,[$, $I(\psi_{min}) \in]-\infty\,;\,\psi_{min}\,[$ is used. However, it takes time to run several SIVIA so it could be interesting to increase the speed of the program.

4.2.2.2.2 First way to increase the speed

To increase the speed of the program, the following method can be implemented: If *m* is not an upper bound, SIVIA will return at least one solution. Thus, the speed of the program can be increased by stopping SIVIA when it finds a solution. Therefore, SIVIA can be modified in a version named MypySIVIA and whenever there is a box inside the solution or undetermined, the execution of MypySIVIA stops. This method increases a lot the speed of the program. For example, to compute the upper bound with this method applied to ψ_1, with $\theta_e\, \epsilon[-\pi\,;\,\pi]$, $u_2\epsilon[-1\,;1]$, $p_1\epsilon[-0.1\,;0.1]$, $u_{2d}\epsilon[-1\,;1]$, $u_1\epsilon[-1\,;1]$, $p_2\epsilon[-0.1\,;0.1]$, it takes seconds.

4.2.2.2.3 Second way to increase the speed

MypySIVIA takes time when there is no solution. For example, if the objective is to find the upper bound, and *m* > ψ_{max} when [*a ; b*] is bisected, MypySIVIA can't be interrupted until the end of the computation of all boxes. To solve this problem, a solution could be to assure that *m* doesn't surpasses too much ψ_{max}. Thus, instead of bisecting the interval [*a ; b*], it can be divided by a variable $\mu \,\epsilon\, \mathbb{N}$ as below:

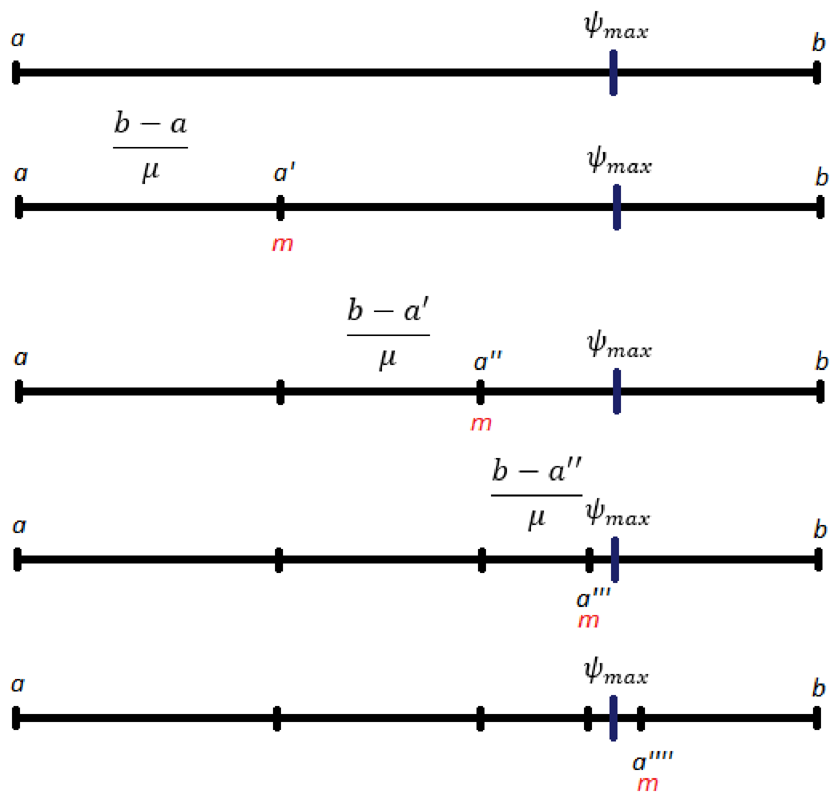


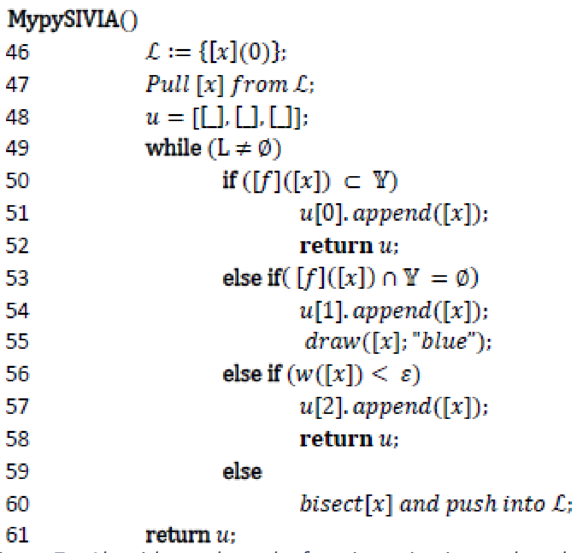
Figure 6 : how to improve the speed of the algorithm

The bigger μ is, the smaller the gap between ψ_{max} and the first value to surpass it, is. In the example above, once ψ_{max} has been surpassed, b is replaced by *a''''*. When the new interval [a''' ; a''''] is smaller than a certain value (named ε_1) the increasing is very small so it takes time to reach ψ_{max} . Therefore, the algorithm stops and starts bisecting by two as previously seen, the new interval [a'''; a'''']. This method is faster for some functions and depends on the value of μ. It reduces the time taken by the program by seconds.

4.2.2.3 Pseudo Code

The figure 7 below describes the algorithm used to find an upper bound and a lower bound to ψ_1 and ψ_2. MaximIVIA() and MinimIVIA() are respectively designed to compute f_{sup} and f_{inf}. To do that, as previously said, a first overestimated approximation is used to proceed as described in "4.2.2.2.3 Second way to increase the speed". FoncSIVIA takes as parameters m, which is the same m as shown in the Figure 5, s which is the parameters taken by ψ, and $bound$ which is a variable equal to 1 when the algorithm computes the upper bound and 0 when it computes the lower bound. The function FoncSIVIA returns a list whose length is equal to three. The first element contains every box inside the solution, the second one every box out of the solution and the third element contains every undetermined box (box whose width is smaller than ε of SIVIA). When there are no solutions and the width of f_{sup} (f_{inf} for MinimIVIA()) is smaller than ε, the program stops and return the upper bound (respectively the lower bound when $bound$ is equal to zero).

```
Main()
        $f_{sup}$ = MaximIVIA() ;
        $f_{min}$ = MinimIVIA() ;

MaximIVIA()
        $f_{sup} = [\psi(s)[0];\ \psi(s)[1]]$ ;
        $bound = 1$;
        while ($f_{sup}[1] - f_{sup}[0] > \varepsilon_1$)
            $m = f_{sup}[0] + \frac{|f_{sup}[1]-f_{sup}[0]|}{\mu}$;
            $u$ = FoncSIVIA($m, s, bound$);
            if ($there\ is\ a\ solution$):
                $f_{sup}[0] = m$ ;
            else
                $f_{sup}[1] = m$ ;
        while ($f_{sup}[1] - f_{sup}[0] > \varepsilon_2$)
            $m = \frac{f_{sup}[1]-f_{sup}[0]}{2}$ ;
            $u$ = FoncSIVIA($m, s, bound$);
            if ($there\ are\ solutions\ or\ there\ are\ undetermined\ boxes$)
                    $f_{sup}[0] = m$ ;
            else if ($there\ is\ no\ undetermined\ box$ )
                    $f_{sup}[1] = m$ ;
        return $[f_{sup}[0]\ ;\ f_{sup}[1]]$;


MinimIVIA()
        $f_{min} = [\psi(s)[0];\ \psi(s)[1]]$ ;
        $bound = 0$;
        while ( $f_{min}[1] - f_{min}[0] > \varepsilon_1$)
            $m = f_{min}[1] - \frac{|f_{sup}[1]-f_{sup}[0]|}{\mu}$;
            $u$ = FoncSIVIA($m, s, bound$);
            if ($there\ is\ a\ solution$):
                $f_{sup}[1] = m$ ;
            else
                $f_{sup}[0] = m$ ;
        while ($f_{min}[1] - f_{min}[0] > \varepsilon_2$)
            $m = \frac{f_{min}[1]-f_{min}[0]}{2}$ ;
            $u$ = FoncSIVIA($m, s, bound$);
            if ($there\ are\ solutions\ or\ there\ are\ undetermined\ boxes$)
                    $f_{min}[1] = m$ ;
            else if($u[2] == []$)
                    $f_{min}[0] = m$ ;
        return $[f_{min}[0]\ ;\ f_{min}[1]]$;

FoncSIVIA($m, s, bound$)
            $I_1 = Interval(m, +\infty)$;
            $I_2 = Interval(-\infty, m)$;
            $X = IntervalVector([s])$;
            if ($bound == 1$)
                $sep\ =\ SepFwdBwd(\psi(s), I_1)$;
            else
                $sep\ =\ SepFwdBwd(\psi(s), I_2)$;
            $u = MypySIVIA(X, sep, \varepsilon)$;
            return $u$;
```



```
MypySIVIA()
    $\mathcal{L} := \{[x](0)\}$;
    $Pull\ [x]\ from\ \mathcal{L}$;
    $u = [[\,],[\,],[\,]]$;
    while ($L \neq \emptyset$)
        if ($[f]([x]) \subset \mathbb{Y}$)
            $u[0].append([x])$;
            return $u$;
        else if( $[f]([x]) \cap \mathbb{Y} = \emptyset$)
            $u[1].append([x])$;
            $draw([x];\text{"blue"})$;
        else if ($w([x]) < \varepsilon$)
            $u[2].append([x])$;
            return $u$;
        else
            $bisect[x]\ and\ push\ into\ \mathcal{L}$;
    return $u$;
```

Figure 7 : Algorithm to bound a function using interval analysis

4.2.3 V-Stability of the system

4.2.3.1 V-stability and Pseudo-code with P = 0.01 and Q = 0.9

Recall: The surface s is defined by:

$$\begin{aligned} s_1 &= \dot{x}_e + x_e \\ s_2 &= \dot{y}_e + y_e \end{aligned}$$

To be V-stable, the following condition has to be true:

$$V(s) \geq 0 \ and \ \dot{V}(s) \leq -\varepsilon$$

With:

- $V(s) = {s_1}^2 + {s_2}^2 - r$
- $\dot{V}(s) = 2\left(s_1(-Qs_1 - Psign(s_1) + [\psi_1]([x_e],[y_e],[\theta_e],[p])) + s_2(-Qs_2 - Psign(s_2) + [\psi_2]([x_e],[y_e],[\theta_e],[p]))\right)$
  $= 2(s_1(-Qs_1 - Psign(s_1) + [-0.5010\ ;\ 0.5010]) + s_2(-Qs_2 - Psign(s_2) + [-0.5010\ ;\ 0.5010]))$

A SIVIA method can be used to identify which s verify those two conditions. The following algorithm shows how to proceed:

```
Main()
    Vdot();
    V();
    UniSep();

Vdot()
    $I = Interval(-\infty, -10^{-9})$;
    seps = [];
    for $p_1$ in arange($p_{1\_lowerbound}, p_{1\_upperbound}, dt$)
        for $p_2$ in arange($p_{2\_lowerbound}, p_{2\_upperbound}, dt$)
            $sep = SepFwdBwd(Vdot, I)$;
            seps.append();
    $sep = seps[0] \cap \ldots \cap seps[n]$;
    return sep;

V()
    $I = Interval(0, \infty)$;
    $sep = SepFwdBwd(V, I)$;
    return sep;

UniSep()
    $sep = Vdot()\ and\ V()$;
    $X = IntervalVector([s])$;
    $u = pySIVIA(X, sep, \varepsilon)$;
```

Figure 8 : "How to plot s that verify both equations $\dot{V} < 0$ and $V \geq 0$"

In the algorithm above, the Vdot function is used to verify the condition $\dot{V} < 0$. The variables $p_1 and\ p_2$ in the function Vdot represent the intervals ψ_1 and ψ_2. Vdot depends of four intervals, s_1, s_2, ψ_1, ψ_2. To plot SIVIA in two dimensions with s_1 in x-axis and s_2 in y-axis, and have a workable graph, it is needed to create two "for" loops for ψ_1, ψ_2. Thus, $\dot{V}$ still depends of ψ_1, ψ_2 but SIVIA only bisects s_1 and s_2 and the graph resulting is in two dimensions. The V function is used to verify the condition $V \geq 0$. Then, the UniSep function is the intersection of the result of Vdot and V functions.

5 – Result

5.1 Boundaries of the functions ψ_1 and ψ_2

After applying the algorithm of the figure 6 to ψ_1 and ψ_2 with, $\theta_e \epsilon[-\pi\,;\,\pi]$, $u_2\epsilon[-1\,;1]$, $p_1\epsilon[-0.1\,;0.1]$, $u_{2d}\epsilon[-1\,;1]$, $u_1\epsilon[-1\,;1]$, $p_2\epsilon[-0.1\,;0.1]$, $\mu = 2$, the boundaries of ψ_1 and ψ_2 are :

$$\psi_1 \in [-0.5010\,;\,0.5010]$$

$$\psi_2 \in [-0.5010\,;\,0.5010]$$

5.2 V-stability with P = 0.01 and Q = 0.9

Now that the bounds of ψ_1 and $\psi_2$are known, the value of s_1 and s_2 for which the system is V-Stable can be computed by applying SIVIA with a separator associated to the constraint $\dot{V}(s) \in]-\infty\,;0]$ and another separator associated to the constraint $V(s) \in [0\,;+\infty[$.

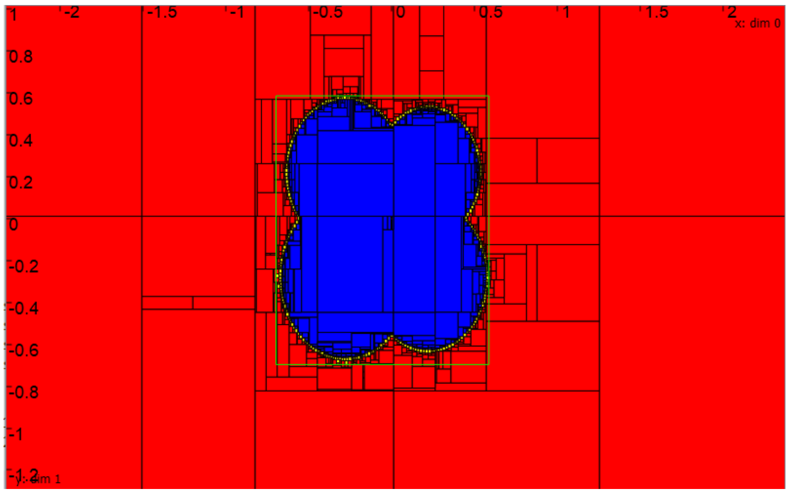


Figure 9 : SIVIA with Q = 0.9, P = 0.01 and r = 0

The red area represents the value of s_1 and s_2 for which the system is V-Stable and the blue area is where the system is not V-stable. The green box whose coordinates are ([-0.6962; 0.5869] ; [-0.6962; 0.5869]) is the smallest box containing the whole blue area. In the figure above that the blue area is very small, thus the area where the system is not V-stable is very small.

5.3 Influence of P and Q on the V-stability

For the figure above, the parameters Q and P are equal to 0.9 and 0.01. The four figures below show the evolution of the blue area with Q and P:

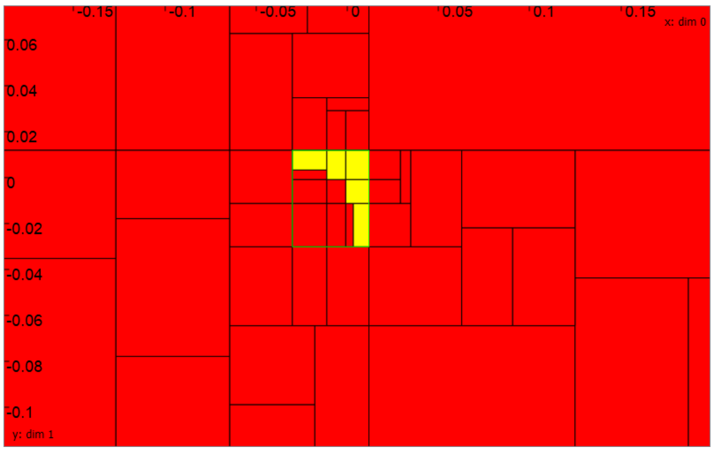


Figure 10:SIVIA with Q = 0.9, P = 0.9 and r =0. Coordinates of the green box: ([-0.0300914, 0.011875] ; [-0.0300914, 0.011875])

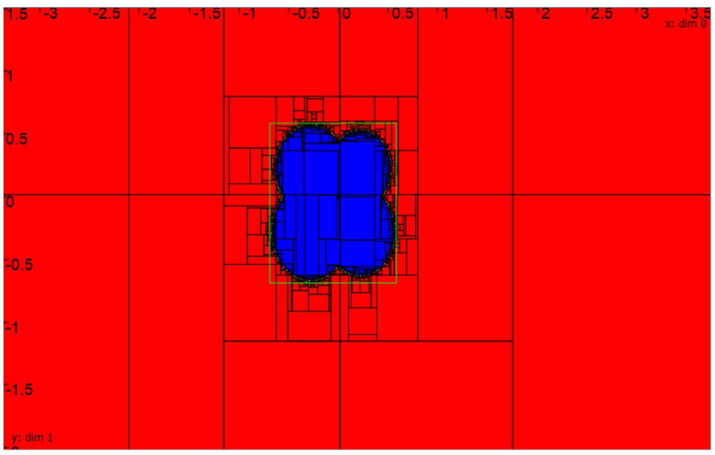


Figure 11: SIVIA with Q = 0.9, P = 0.01 and r = 0. Coordinates of the green box: ([-0.68782, 0.585286] ; [-0.688347, 0.585286])

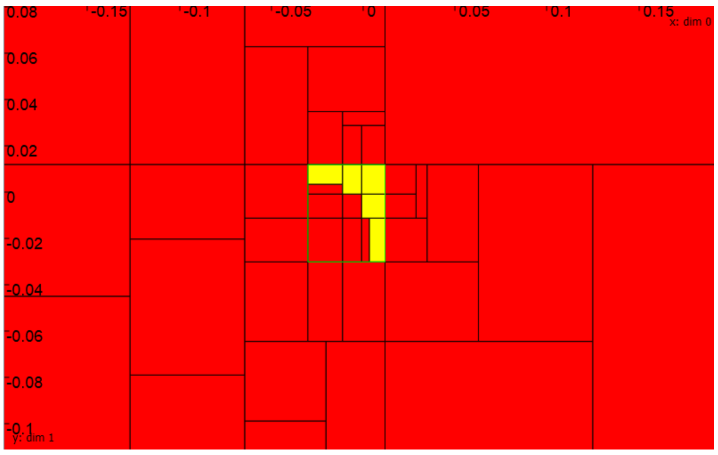


Figure 12: SIVIA with Q = 0.01, P = 0.9 and r =0. Coordinates of the green box: ([-0.0300914, 0.011875] ; [-0.0300914, 0.011875])

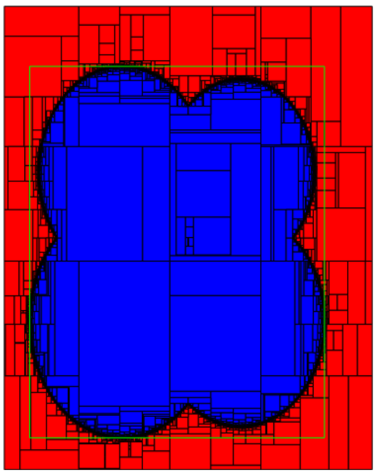


Figure 13 :SIVIA with Q = 0.01, P = 0.01 and r = 0. Coordinates of the green box: ([-60.3669, 51.8616] ; [-60.3669, 51.8616])

When P is high ($P = 0.9$), changing Q has no influence on the blue area, it keeps the same size (referring to the figure 10 and 12). However, when Q is equal to 0.9, changing P modifies the V-Stability zone, as shown in the figure 13 it becomes smaller. If P is small ($P = 0.01$) and Q is as smaller, the blue area becomes very big as seen in the figure 11.

If Q is near 1, then reducing P a lot has not a considerable influence on the size of the blue area as seen below:

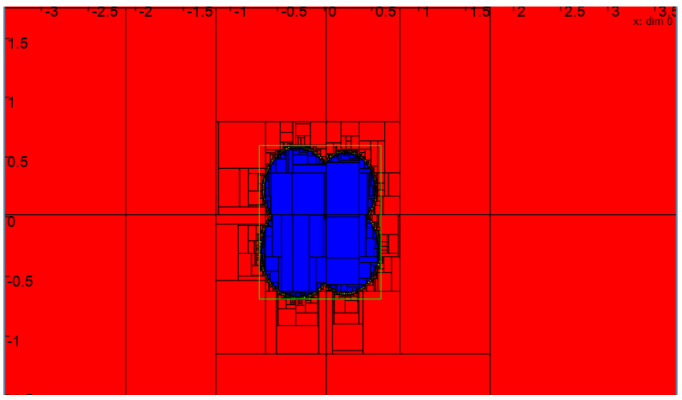


Figure 14: SIVIA with Q = 0.9, P = 0.0001 and r =0. Coordinates of the green box: ([-0.69168, 0.598091] ; [-0.691529, 0.598091])

Those figures show that the smaller P is, the bigger the blue area is. Despite this fact, this area remains tiny even if P is very small (= 0.0001) as above. Nevertheless, the size of the blue area increases rapidly when we reduce Q. So, when Q reduces and P is small, the system is much less stable.

5.4 Application of this algorithm to other surfaces.

5.4.1 Definition of two others surfaces

Previously the sliding mode controller was applied to the surface:

$$S_1 = \begin{pmatrix} \dot{x_e} + x_e \\ \dot{y_e} + y_e \end{pmatrix}$$

Consider now the sliding mode with other surfaces:

$$S_2 = \begin{pmatrix} \dot{x_e} + k_1 x_e \\ \dot{y_e} + k_2 y_e + k_0 sign(y_e).|\theta_e| \end{pmatrix}$$

And

$$S_3 = \begin{pmatrix} \dot{x_e} + k_1 x_e \\ \dot{\theta_e} + k_2 \theta_e + k_0 y_e \end{pmatrix}$$

For the following computations, the values of the several parameters will be $\theta_e \in [-\pi\,;\,\pi]$, $u_2 \in [-1\,;1]$, $p_1 \in [-0.1\,;0.1]$, $u_{2d} \in [-1\,;1]$, $u_1 \in [-1\,;1]$, $p_2 \in [-0.1\,;0.1]$, $k_0 = 0.6, k_1 = 0.5, k_2 = 0.8$, $\mu = 8$, P = 0.01, Q = 0.9

5.4.2 V-stability of S_2

The dynamic for the error is still the same:

$$\dot{s} = -Qs - Psign(s) + \psi(x_e, y_e, \theta_e, p)$$

Thus, $\psi(x_e, y_e, \theta_e, p)$ must be computed

Recall:

$$\begin{pmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{pmatrix} = \begin{pmatrix} -u_{1d} + u_1 \cos\theta_e + u_{2d} y_e \\ u_1 \sin\theta_e - u_{2d} x_e \\ u_2 - u_{2d} \end{pmatrix}$$

Expression of $\dot{s_1}$ and $\dot{s_2}$:

$$\dot{s_1} = \dot{u_1}\cos(\theta_e) + k_1(-u_{1d} + (u_1 + p_1)\cos(\theta_e) + y_e u_{2d}) + \dot{u_{2d}} y_e + u_{2d}\big((u_1 + p_1)\sin(\theta_e) - x_e u_{2d}\big) - (u_1 + p_1)(u_2 + p_2 - u_{2d})\sin(\theta_e) - \dot{u_{1d}}$$

$$\dot{s_2} = (u_2 + p_2 - u_{2d})((u_1 + p_1)\cos(\theta_e) + k_0 sign(y_e)) + k_2\left((u_1 + p_1)\sin(\theta_e) - x_e u_{2d}\right) + \dot{u_1}\sin(\theta_e) - \dot{u_{2d}} x_e - u_{2d}(-u_{1d} + (u_1 + p_1)\cos(\theta_e) + y_e u_{2d})$$

By suppressing the terms which do not contain p_1 or p_2, then:

$$\psi_1 = \cos(\theta_e)\,(k_1 p_1) + \sin(\theta_e)(p_1 u_{2d} - u_1 p_2 - p_1 u_2 - p_1 p_2 + p_1 u_{2d})$$

$$\psi_2 = p_1 \cos(\theta_e)\,(u_2 + p_2 - 2u_{2d}) + \sin(\theta_e)(k_2 p_1)$$

The functions ψ_1 and ψ_2 can be bounded with MypySIVIA.

$$\psi_1 \epsilon \ [-0.4131\ ;\ 0.4131]$$

$$\psi_2 \epsilon \ [-0.3223\ ;\ 0.3223]$$

ψ_1 and ψ_2 are bounded so now the V-stability can be studied.

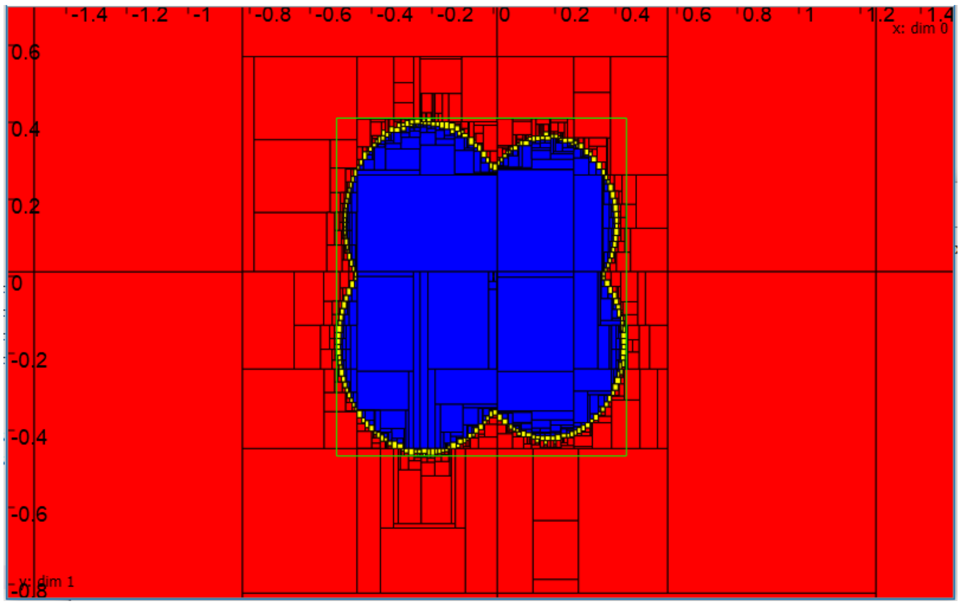


Figure 15 : SIVIA to the 2nd surface with Q = 0.9, P = 0.01 and r = 0

The size of the green box is ([-0.5142, 0.4354] ; [-0.4663, 0.4107]). The blue area has the same shape as the first surface, however in this case, the blue area is smaller. Thus, the sliding mode applied to the second surface is more robust than for the first surface.

5.4.3 V-stability of S_3

<u>*Recall*</u>:

$$S_3 = \begin{pmatrix} \dot{x_e} + k_1 x_e \\ \dot{\theta_e} + k_2\theta_e + k_0 y_e \end{pmatrix}$$

$$\begin{pmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{pmatrix} = \begin{pmatrix} -u_{1d} + u_1\cos\theta_e + u_{2d}y_e \\ u_1\sin\theta_e - u_{2d}x_e \\ u_2 - u_{2d} \end{pmatrix}$$

Expression of $\dot{s_1}$ and $\dot{s_2}$:

$$\dot{s_1} = \dot{u_1}\cos(\theta_e) + k_1(-u_{1d} + (u_1+p_1)\cos(\theta_e) + y_e u_{2d}) + \dot{u_{2d}}y_e + u_{2d}\big((u_1+p_1)\sin(\theta_e) - x_e u_{2d}\big) - (u_1+p_1)(u_2+p_2-u_{2d})\sin(\theta_e) - \dot{u_{1d}}$$

$$\dot{s_2} = \dot{u_2} + k_0((u_1+p_1)\sin(\theta_e) - x_e u_{2d}) - \dot{u_{2d}} + k_2(\dot{u_2} - \dot{u_{2d}})$$

After suppressing the terms without p_1 or p_2:

$$\psi_1 = \cos(\theta_e)\,(k_1 p_1) + \sin(\theta_e)(p_1 u_{2d} - u_1 p_2 - p_1 u_2 - p_1 p_2 + p_1 u_{2d})$$

$$\psi_2 = k_0 p_1 \sin(\theta_e)$$

Thus,

$$\psi_1 \epsilon\ [-0.4131\ ;\ 0.4131]$$

$$\psi_2 \epsilon\ [-0.06\ ;\ 0.06]$$

V-stability of the surface S_3:

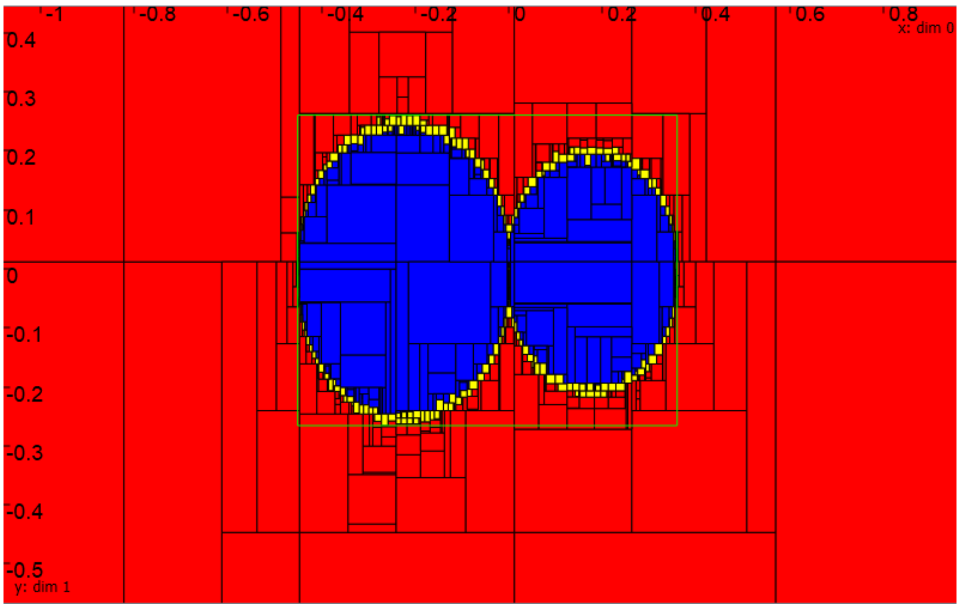


Figure 16 : SIVIA to the 3rd surface with Q = 0.9, P = 0.01 and r = 0

The size of the green box is ([-0.450872, 0.359851] ; [-0.265748, 0.260447]). The shape of the blue area has changed and the green box is smaller than any previous surfaces. This means that sliding mode applied to this surface is more robust than the other surfaces.

6 - Analysis of the sliding mode result

According to the results above, even with uncertainties $p_1 \epsilon [-0.1 ; 0.1]$, $p_2 \epsilon [-0.1 ; 0.1]$, the sliding mode is robust. However, the blue area where the system is no longer stable created by those uncertainties, depends also of the parameters P and Q of the equation:

$$\dot{s} = -Qs - Psign(s) + \psi(x_e, y_e, \theta_e, p)$$

The bigger P and Q are, the smaller the blue area will be. Nevertheless, those parameters also affect the behaviour of the sliding mode controller. If the value of Q is high, the error will converge faster to the surface where it is null. The value of P affects the shattering. The bigger P is, the more shattering there will be, thus it is important to reduce P as much as possible otherwise the system could not be stable because it would keep shattering.

A first solution to keep the system stable and to avoid too much shattering, could be to put a high value for Q (0.9) and a low value for P. Thus, as shown in the figures 10 and 14, the blue area is still small, and the shattering should be very low.

However, the figure 10 and 12 show something that should not happen. Indeed, between those two figures, only Q changes. This parameter which is multiplied by s in the dynamic of the error, is the one that should affect the most the size of the unsafe area. Effectively, Q has a direct impact on the time it takes for the error to be equal to zero. Yet, the unsafe area is not affected. A first explanation could be that the term $Psign(s)$ compensated the low value of Q. The figure 11 and 13 could confirm this theory because they show that with a low P $(= 0.01)$, changing Q multiplied by approximatively 1 000 the surface of the green box.

Otherwise the results seem to agree the theory, when Q and P decrease, the safe area decrease and equivalently when they increase, it takes less time to reach the sliding mode surface so the safe area increase.

A second solution to keep the system stable could be to change the sliding mode surface. Indeed, the figure 16 shows that the unstable area is smaller than the figure 10, even though the same parameters are used for both surfaces.

Another point of interest is the shape of the unsafe area in the figures 11, 13, 14, 15, 16. The shape of it looks like a combination of circles. According to the differences between figure 15 and 16, the shape depends of the equation of the sliding mode surface. Unfortunately, I didn't have the time to investigate in this subject and to give a correct explanation. At my point, I can only note that the shape is affected by the sliding mode surface and Q and P.

To resume:

- If P decreases: the shattering decreases and the area where the system is V-stable reduces but stays important if Q is big (≈ 1).
- If Q decreases: the system converges slowly to the surface s. If P is great (≈ 1) the area where the system is V-stable does not change. However, if P is small, this area reduces a lot.
- The stable area is affected by the sliding mode surface.

All in all, despite uncertainties in the state equations of the robot, the sliding mode is still robust. This algorithm is a tool that give information about the link between the sliding mode surface, Q and P and the stable area.

7 - Feedback linearization with uncertainties in the state equation

7.1 Introduction

During my internship, I was asked to determine the safe area of the following system:

$$\begin{cases} \dot{x} = x^3(1+p_1) + xu \\ y = \exp(x+p_2) \end{cases}$$

With $p_i \in [-0.1\,;0.1]$ but instead of applying sliding mode, a feedback linearization is applied. To do a feedback linearization, y must be derived until the input u appears in the equation.

$$\dot{y} = x^3(1+p_1)\exp(x+p_2) + xexp(x+p_2)u$$

Thus,

$$u = \frac{1}{xexp(x)}(-x^3\exp(x) + v)$$

With v the new input. Then $\dot{y}$ becomes:

$$\dot{y} = x^3(1+p_1)\exp(x+p_2) + \exp(p_2)(-x^3\exp(x)+v)$$
$$= \underbrace{x^3 p_1 \exp(p_2)\exp(x)}_{b(p,x)} + \underbrace{\exp(p_2)}_{A(p,x)} v$$

Take a proportional control $v = -y$, then $\dot{y} = -A(p, \ln y - p_2).y + b(p, \ln y - p_2)$

$$\dot{y} = -\exp(p_2)\, y + (\ln y - p_2)^3 y.p_1$$

7.2 V-Stability of the feedback linearization control

Consider the following differential inclusion:

$$V(y) = y^2 - r$$

Recall: to prove the V-stability of a system, the next condition should be checked:

$$V(x) \geq 0 \implies \dot{V}(x) \leq -\varepsilon$$

With $\varepsilon > 0$.

Here $\dot{V}(y) = 2y\dot{y} = 2y(-\exp(p_2)\,.y + (\ln y - p_2)^3.y.p_1)$

To verify the V-stability condition, the same algorithm as the figure 8 can be used. By replacing y by $y = e^{x+p2}$, $V(x,p)$ and $\dot{V}(x,p)$ become:

$$V(x,p) = e^{2(x+p_2)} - r$$

$$\dot{V}(x,p) = 2e^{x+p_2}(-e^{p_2}.e^{x+p_2} + x^3.y.p_1)$$

7.3 Result

The figure 17 below shows a SIVIA with a separator associated to the constraint $\dot{V}(x,p) \in]-\infty\, ; -\varepsilon]$ with $\varepsilon \approx 0$ (To get a workable graph and a correct size of the axis, $p_1 \in [-10; 10], p_2 \in [-10; 10]$ and $x \in [-10; 10]$)

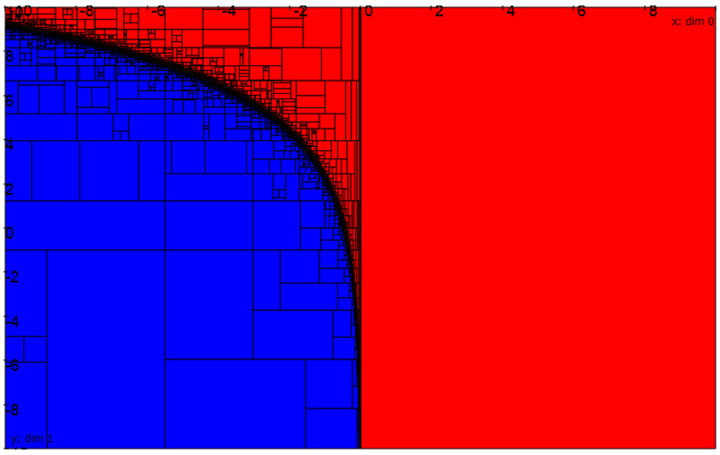


Figure 17: SIVIA with x in x-axis and p2 in y-axis. The clearer area verify V̇(x,p) <0

The figure 18 shows a SIVIA with a separator associated to the constraint $V(x,p) \in [\,0\,; +\infty[$ with $r = 2$:

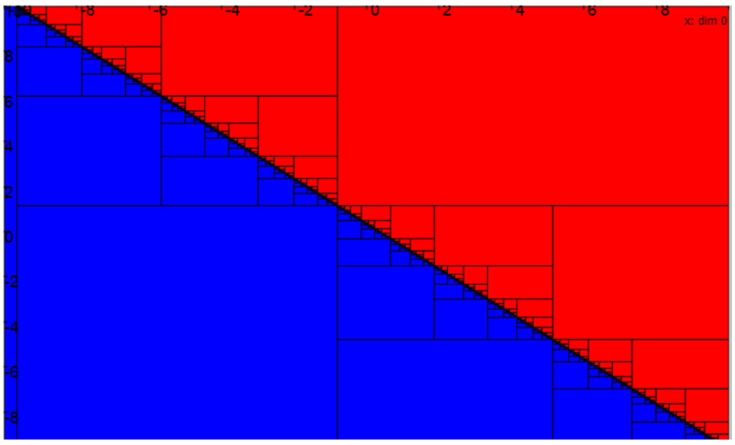


Figure 18: SIVIA with x in x-axis and p2 in y-axis. The clearer area verify V(x,p)≥0 with r=2

The figure 19 shows the intersection between the figure 17 and the figure 18:

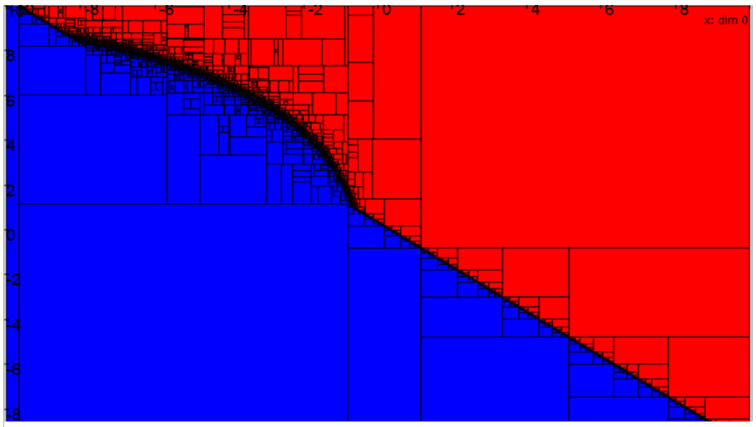


Figure 19: Intersection of figure 17 and figure 18

The clearer (red) area in the figure 19 is where both equations of the V-stability approach are verified, so where the system is V-stable.

7.4 Analysis of the result

Thanks to the same algorithm as previously, is it possible to plot the safe area of this system. Nevertheless, an interesting point could be to compare the robustness of the feedback linearization and the sliding mode against uncertainties. However, here the state equation is not the same as the one taken in the part 4. so, there is nothing to compare. The state equation of the part 7. was just given as an example, so I decided to apply feedback linearization to the following equation:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} u_1 \cos\theta \\ u_1 \sin\theta \\ u_2 \end{pmatrix}$$

in order to get something comparable.

7.5 Comparison between Sliding mode and feedback linearization

Firstly, as previously seen, by doing the same computation as in the part 4.1:

$$\begin{pmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{pmatrix} = \begin{pmatrix} -u_{1d} + u_1 \cos\theta_e + u_{2d} y_e \\ u_1 \sin\theta_e - u_{2d} x_e \\ u_2 - u_{2d} \end{pmatrix}$$

Then u_{1d} and u_{2d} can be deduced:

$$\begin{pmatrix} u_{1d} \\ u_{2d} \end{pmatrix} = \begin{pmatrix} -1 & y_e \\ 0 & -x_e \end{pmatrix}^{-1} \left(\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} - \begin{pmatrix} u_1 cos\theta_e \\ u_1 sin\theta_e \end{pmatrix}\right)$$

$$\begin{cases} u_{1d} = -(v_1 - u_1 cos\theta_e) - \frac{y_e}{x_e}(v_2 - u_1 sin\theta_e) \\ u_{2d} = -\frac{1}{x_e}(v_2 - u_1 sin\theta_e) \end{cases}$$

By replacing u_{1d} and u_{2d} in the expression of $\dot{x_e}$ and $\dot{y_e}$ and adding the uncertainties $p_1$and $p_2$to the terms $u_1$and u_2 we obtain:

$$\begin{cases} \dot{x_e} = v_1 - p_1 cos\theta_e \\ \dot{y_e} = v_2 + p_1 sin\theta_e \end{cases}$$

After taking a proportional control $v_1 = -x_e$ and $v_2 = -y_e$ the system becomes:

$$\begin{cases} \dot{x_e} = -x_e - p_1 cos\theta_e \\ \dot{y_e} = -y_e + p_1 sin\theta_e \end{cases}$$

To study the V-stability, the same differential inclusion as in the 4.2.1 section is taken: $V(s) = s^T s - r$. It implies that:

$$V(x,p) = {x_e}^2 + {y_e}^2 - r$$

$$\dot{V}(x,p) = 2x_e(-x_e - p_1 cos\theta_e) + 2y_e(-y_e + p_1 sin\theta_e)$$

In order to plot the V-stability and to compare with the sliding, the same parameters will be taken ($\theta_e \in [-\pi\ ;\ \pi]$, $p_1 \in [-0.1\ ;0.1]$, $x_e \in [-10\ ;10]$ and $y_e \in [-10\ ;10]$)
The figure below shows the result:

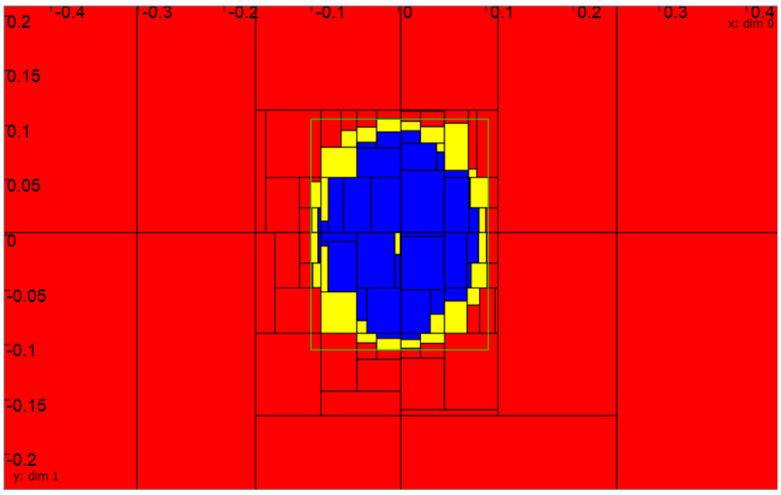


Figure 20: V-stability area in red, xe in the x-axis, ye in the y-axis. Coordinates of the green box :([-0.100803, 0.102846] ; [-0.104555, 0.105737])

The figure 20 shows that the feedback linearization is robust against uncertainties, the size of the green box is small. To compare with the sliding mode, according to the figure 10 if the P and Q of the dynamic of the error are equal to 0.9, the sliding mode is safer than feedback linearization (referring to figure 10 and 20). However, it is no longer the case when the value of P is low as the figures 11, 13, 14 show. All in all, if the sliding mode control is correctly designed, it is more robust against uncertainties than the feedback linearization.

8 - What did the internship bring to me

This internship was for me a first opportunity to apply my knowledge and skills learned at ENSTA Bretagne. Indeed, this traineeship gave me the chance to put in practice my knowledge about interval analysis and to improve it through the projects carried out. This period also gave me a first experience in the working world as an assistant engineer, to get a first approach of the rhythm of a research engineer/PhD student, which consists of learning new theories, working on projects, presenting the progress in the project to the supervisor, giving lessons etc…During this internship, I had faced technical problem (such as program that return a result not accurate enough, which takes a long time to run…) which made me working on how to solve those kinds of problems. This goes through the search of new theories that could lead to a solution, through thinking of new methods more efficient… As an example, bounding the functions ψ_1 and ψ_2 (functions containing the uncertainties of the system) led to some problems. Effectively, the first version of my algorithm returned an inaccurate result which took time to be computed (several decades of minutes). To improve the accuracy and the speed of the program I rethought the algorithm and eliminated the cases where the computer deals with useless computations. To be confronted to problems, to rethink the algorithm in order to improve the efficiency and to reach a solution was for me a very interesting experience. This internship enabled me to get a first understanding of the research world, my Final Study Project will be the opportunity to discover the company world as an engineer and thus to define the sector I want to work in later.

Unfortunately, I do not have the feedback paper from my supervisor, I knew about this paper when I was back in France. I sent it to my supervisor by mail but I have not received an answer yet…

9 - Conclusion

To conclude, this report shows a way to display and to quantify the stable area of a system subject to uncertainties and regulated either by a sliding mode control or a feedback linearization. To obtain this result with the sliding mode control, the equation of the sliding mode surface with uncertainties must be computed. Then, the terms with the uncertainties, named ψ_1 and ψ_2, in the last equation must be bounded in order to determine their influence on the dynamic of the error. After adopting an interval analysis approach and thinking of a new way to bound those functions quickly and accurately, the MypySIVIA had been created and enabled the bounding of ψ_1 and ψ_2. Thus, the area where the V-stability conditions are consistent was computable. After that, a SIVIA method was enough to verify the V-stability conditions and to display the safe area. The dynamic of the error was $-Qs - Psign(s) + \psi(x_e, y_e, \theta_e, p)$ so several plots had been necessary to evaluate the influence of Q and P on the V-stability. Those two parameters impact the size of the stable area but also the way the robot behave. A $Q \approx 1$ and $P \approx 0.01$ leads to a small unstable area and a low shattering effect. The equation of the surface has an impact too on the stable area as seen previously, but no explanation to link the sliding mode surface and the shape of the stable area has been done yet. About the feedback linearization, the regulation is safer than the sliding mode when the values of Q and P are low. However, for both regulations, the unstable area is small, it can be concluded that both regulations are robust against uncertainties.

Concerning my personal project, this internship gave me a first experience of how working in the research sector is like, I could put my skill in practice and to develop them through the carried projects. This traineeship was very interesting for my personal project because before being graduated from the school I want to get an experience in the research sector and the company sector, in order to clarify in which sector I would like to work in later.

10 - Future Work

To continue the work, it could be interesting to determinate the link between the parameters Q and P and the shape of the unstable area, the same with the sliding mode surface used. By doing this, it could be possible to choose a surface which minimize the risk of instability and to determine optimal value for Q and P.

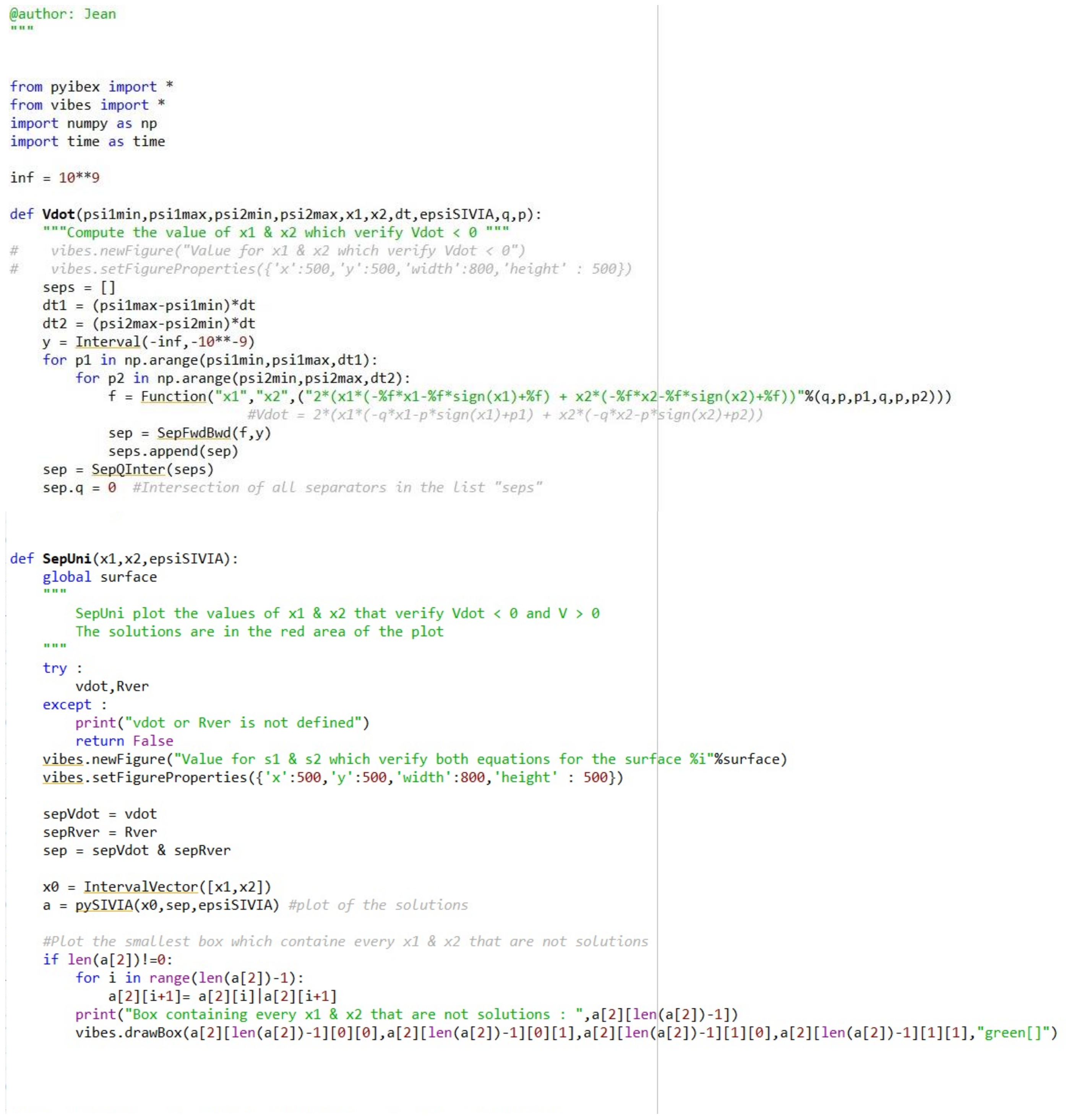
11 – References

[1] L. JAULIN, F. L. BARS. “An interval approach for stability analysis; Application to sailboat robotics”. ENSTA Bretagne (2012)

[2] A. STANCU, L. JAULIN. “Set-membership tracking using interval analysis”.

12 – Tables of figures

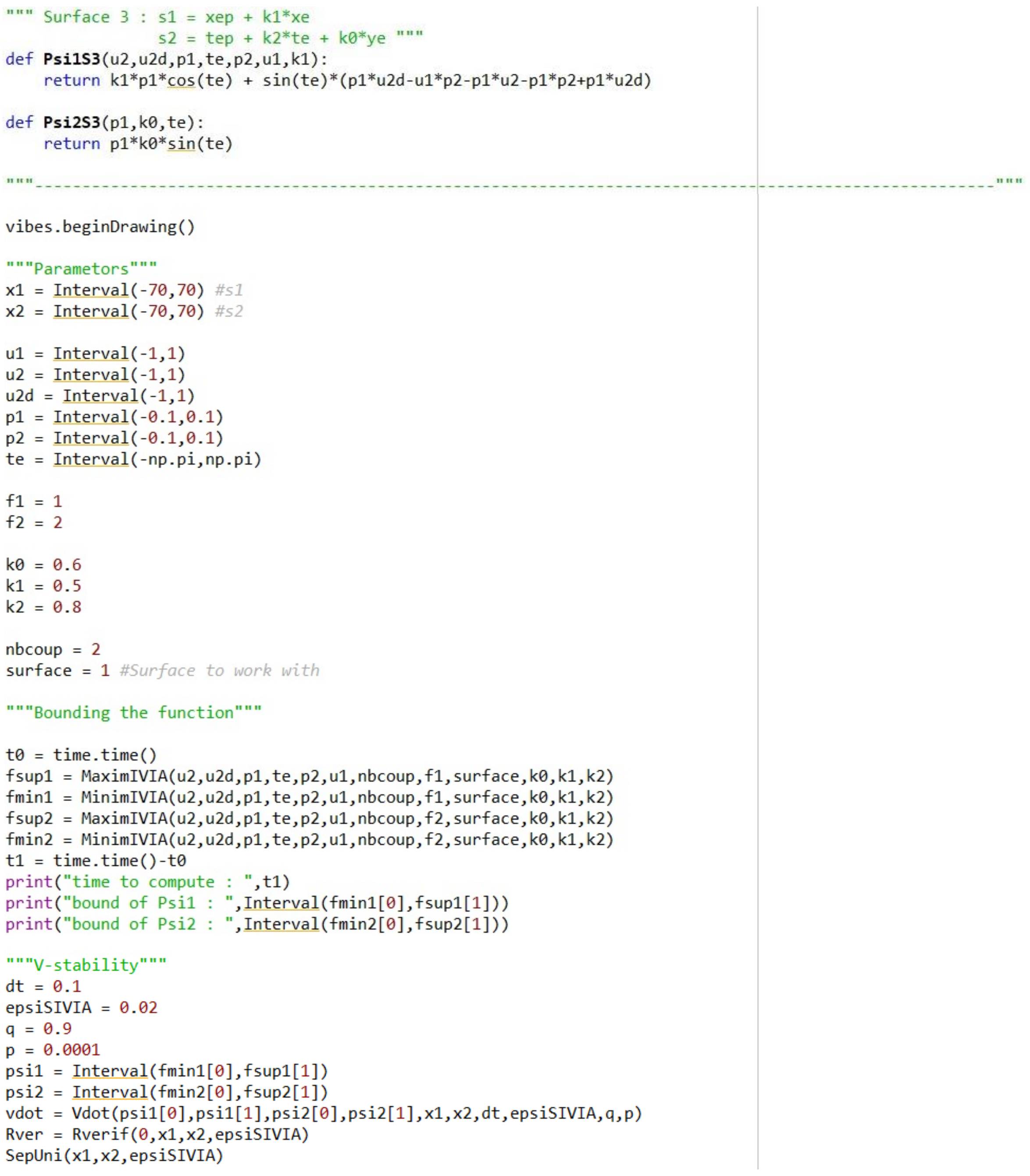
13 - Appendices



```
@author: Jean
"""


from pyibex import *
from vibes import *
import numpy as np
import time as time

inf = 10**9

def Vdot(psi1min,psi1max,psi2min,psi2max,x1,x2,dt,epsiSIVIA,q,p):
    """Compute the value of x1 & x2 which verify Vdot < 0 """
#    vibes.newFigure("Value for x1 & x2 which verify Vdot < 0")
#    vibes.setFigureProperties({'x':500,'y':500,'width':800,'height' : 500})
    seps = []
    dt1 = (psi1max-psi1min)*dt
    dt2 = (psi2max-psi2min)*dt
    y = Interval(-inf,-10**-9)
    for p1 in np.arange(psi1min,psi1max,dt1):
        for p2 in np.arange(psi2min,psi2max,dt2):
            f = Function("x1","x2",("2*(x1*(-%f*x1-%f*sign(x1)+%f) + x2*(-%f*x2-%f*sign(x2)+%f))"%(q,p,p1,q,p,p2)))
                                   #Vdot = 2*(x1*(-q*x1-p*sign(x1)+p1) + x2*(-q*x2-p*sign(x2)+p2))
            sep = SepFwdBwd(f,y)
            seps.append(sep)
    sep = SepQInter(seps)
    sep.q = 0  #Intersection of all separators in the list "seps"



def SepUni(x1,x2,epsiSIVIA):
    global surface
    """
        SepUni plot the values of x1 & x2 that verify Vdot < 0 and V > 0
        The solutions are in the red area of the plot
    """
    try :
        vdot,Rver
    except :
        print("vdot or Rver is not defined")
        return False
    vibes.newFigure("Value for s1 & s2 which verify both equations for the surface %i"%surface)
    vibes.setFigureProperties({'x':500,'y':500,'width':800,'height' : 500})

    sepVdot = vdot
    sepRver = Rver
    sep = sepVdot & sepRver

    x0 = IntervalVector([x1,x2])
    a = pySIVIA(x0,sep,epsiSIVIA) #plot of the solutions

    #Plot the smallest box which containe every x1 & x2 that are not solutions
    if len(a[2])!=0:
        for i in range(len(a[2])-1):
            a[2][i+1]= a[2][i]|a[2][i+1]
        print("Box containing every x1 & x2 that are not solutions : ",a[2][len(a[2])-1])
        vibes.drawBox(a[2][len(a[2])-1][0][0],a[2][len(a[2])-1][0][1],a[2][len(a[2])-1][1][0],a[2][len(a[2])-1][1][1],"green[]")
```

```python
def FoncSIVIA(fsup,u2,u2d,p1,te,p2,u1,fPsi,surface,bound,k0,k1,k2):
    """
    This function applies SIVIA to find the upper bound or the lower bound of a function
    FoncSIVIA[0] return the SIVIA used to find the upper bound
    FoncSIVIA[1] return the SIVIA used to find the lower bound
    fPsi determines which equation of Psi will be used for the SIVIA
    (fPsi = 1 for the first equation, fPsi = 2 for the second equation)
    surface determines the surface used
    bound determines if we want the lower bound or the upper bound
    (1 for the upper bound, 0 for the lower bound)
    """

    vibes.newFigure("FoncIVIA")
    vibes.setFigureProperties({'x':500,'y':500,'width':800,'height' : 500})

    I = Interval(fsup,inf) #Interval used to find the upper bound
    I2 = Interval(-inf,fsup) #Interval used to find the lower bound

    if surface == 1:
        epsIVIA = 0.0005 #Value of the epsilon used for SIVIA
        x0 = IntervalVector([u2,u2d,p1,te,p2,u1])
        if fPsi == 1 :
            f = Function("u2","u2d","p1","te","p2","u1","p1*(-(u2-u2d)*sin(te)+u2d*sin(te)+u2d*sin(te)+cos(te))-p2*sin(te)*u1")
        elif fPsi == 2 :
            f = Function("u2","u2d","p1","te","p2","u1","p1*((u2-u2d)*cos(te)-u2d*cos(te)-u2d*cos(te)+sin(te))+p2*u1*cos(te)")

    elif surface == 2 :
        x0 = IntervalVector([u2,u2d,p1,te,p2,u1])
        if fPsi == 1 :
            epsIVIA = 0.0005
            f = Function("u2","u2d","p1","te","p2","u1","%f*p1*cos(te) + sin(te)*(p1*u2d-u1*p2-p1*u2-p1*p2+p1*u2d)"%k1)
        elif fPsi == 2 :
            epsIVIA = 0.05
            f = Function("u2","u2d","p1","te","p2","u1","p1*cos(te)*(u2+p2-2*u2d)+sin(te)*(%f*p1)"%k2)

    elif surface == 3 :
        epsIVIA = 0.0005
        if fPsi == 1 :
            x0 = IntervalVector([u2,u2d,p1,te,p2,u1])
            f = Function("u2","u2d","p1","te","p2","u1","%f*p1*cos(te) + sin(te)*(p1*u2d-u1*p2-p1*u2-p1*p2+p1*u2d)"%k1)
        elif fPsi == 2 :
            x0 = IntervalVector([p1,te])
            f = Function("p1","te","p1*%f*sin(te)"%k0)

    if bound == 0:
        sep = SepFwdBwd(f,I2)
    else :
        sep = SepFwdBwd(f,I)
    a = MypySIVIA(x0,sep,epsIVIA) #MypySIVIA is a SIVIA That stops when there is one box inside the solution or undetermined

    return a



def MaximIVIA(x1,x2,p1,p2,p2b,u1,nbcoup,fPsi,surface,k0,k1,k2):
    """
    MaximIVIA returns the upper bound of the function
    nbcoup determines in how much the interval is split at first
    surface determines the surface used
    fPsi determines which equation of Psi will be used for the SIVIA
    (fPsi = 1 for the first equation, fPsi = 2 for the second equation)
    """
    if surface == 1:
        if fPsi == 1 :
            u = Psi1S1(x1,x2,p1,p2,p2b,u1)
        elif fPsi == 2 :
            u = Psi2S1(x1,x2,p1,p2,p2b,u1)

    elif surface == 2 :
        if fPsi == 1 :
            u = Psi1S2(u2,u2d,p1,te,p2,u1,k1)
        elif fPsi == 2 :
            u = Psi2S2(u2,u2d,p1,te,p2,u1,k2)

    elif surface == 3 :
        if fPsi == 1 :
            u = Psi1S3(u2,u2d,p1,te,p2,u1,k1)
        elif fPsi == 2 :
            u = Psi2S3(p1,k0,te)

    bound = 1 #if we want to find the upper bound, bound has to be equal to 1
    fsup = [u[0],u[1]] #First approximation of the bounds.
    epsilon = 10**(np.log10(np.abs(fsup[1]))//1-3)  #Accuracy of the bound. The result will be an interval
                                                    #which width is 10^k*10^-3 with k equal to the exponent of the solution
                                                    #(Exemple if the solution is 0.04, epsilon will be equal to 10^-2*10^-3 = 10^-5)

    fint = fsup[0]
    while fsup[1]- fint > (2*nbcoup*10**(np.log10(np.abs(fsup[1]))//1-2))+1 : #Split the interval in nbcoup parts until the size of
                                                                                #the final interval is 2*nbcoup*(10^-2*10^k)+1
                                                                                #This method is designed to reach a good interval quickly


        test = 1
        while test == 1 :
            m = fsup[0]
            a = FoncSIVIA(m,x1,x2,p1,p2,p2b,u1,fPsi,surface,bound,k0,k1,k2)
            if (a[2] == [] and a[0]!= []) or a[2]!=[]:
                fint = fsup[0]
                fsup[0] += np.abs(fsup[1]-fsup[0])/nbcoup
            elif a[2] == []:
                test = 0

        fsup[1] = fsup[0]
        fsup[0] = fint
```

```
  while fsup[1]-fsup[0] > epsilon : #Once the interval of the solution is smaller than 2*nbcoup*(10^-2*10^k)+1,
                                    # we bissect the interval to reach the solution quickly
      m = (fsup[0]+fsup[1])/2
      a = FoncSIVIA(m,x1,x2,p1,p2,p2b,u1,fPsi,surface,bound,k0,k1,k2)

      if (a[2] == [] and a[0]!= []) or a[2]!=[]: #If there is a solution in the interval
                                                 #[m,fsup[1]], then fsup[0]=m, else if there is no solution fsup[1] = m
          fsup[0] = m
      elif a[2] == [] :
          fsup[1] = m

  return [fsup[0],fsup[1]]


def MinimIVIA(x1,x2,p1,p2,p2b,u1,nbcoup,fPsi,surface,k0,k1,k2):
    """
    The same function as MaximIVIA but to find the lower bound of the function
    """
    if surface == 1:
        if fPsi == 1 :
            u = Psi1S1(x1,x2,p1,p2,p2b,u1)
        elif fPsi == 2 :
            u = Psi2S1(x1,x2,p1,p2,p2b,u1)

    elif surface == 2 :
        if fPsi == 1 :
            u = Psi1S2(u2,u2d,p1,te,p2,u1,k1)
        elif fPsi == 2 :
            u = Psi2S2(u2,u2d,p1,te,p2,u1,k2)

    elif surface == 3 :
        if fPsi == 1 :
            u = Psi1S3(u2,u2d,p1,te,p2,u1,k1)
        elif fPsi == 2 :
            u = Psi2S3(p1,k0,te)

    bound = 0 #if we want to find the lower bound, bound has to be equal to 0
    fmin = [u[0],u[1]]
    epsilon = 10**(np.log10(np.abs(fmin[1]))//1-3)
    fint = fmin[1]
    while fint- fmin[0] > (2*nbcoup*10**(np.log10(np.abs(fmin[1]))//1-2))+1 :
            test = 1
            while test == 1 :
                m = fmin[1]
                a = FoncSIVIA(m,x1,x2,p1,p2,p2b,u1,fPsi,surface,bound,k0,k1,k2)
                if (a[2] == [] and a[0]!= []) or a[2]!=[]:
                    fint = fmin[1]
                    fmin[1] -= np.abs(fmin[1]-fmin[0])/nbcoup
                elif a[2] == []:
                    test = 0

            fmin[0] = fmin[1]
            fmin[1] = fint

    while fmin[1]-fmin[0]> epsilon :
        m = (fmin[0]+fmin[1])/2
        a = FoncSIVIA(m,x1,x2,p1,p2,p2b,u1,fPsi,surface,bound,k0,k1,k2)

        if (a[2] == [] and a[0]!= []) or a[2]!=[]:
            fmin[1] = m
        elif a[2] == [] :
            fmin[0] = m

    return [fmin[0],fmin[1]]

""" Surface 1 : s1 = xep + xe
                s2 = yep + ye """
def Psi1S1(u2,u2d,p1,te,p2,u1):
    return p1*(-(u2-u2d)*sin(te)+u2d*sin(te)+u2d*sin(te)+cos(te))-p2*sin(te)*(u1+p1)

def Psi2S1(u2,u2d,p1,te,p2,u1):
    return p1*((u2-u2d)*cos(te)-u2d*cos(te)-u2d*cos(te)+sin(te))+p2*cos(te)*(u1+p2)

""" Surface 2 : s1 = xep + k1*xe
                s2 = yep + k2*ye + k0*sign(ye)*|te| """
def Psi1S2(u2,u2d,p1,te,p2,u1,k1):
    return k1*p1*cos(te) + sin(te)*(p1*u2d-u1*p2-p1*u2-p1*p2+p1*u2d)

def Psi2S2(u2,u2d,p1,te,p2,u1,k2):
    return p1*cos(te)*(u2+p2-2*u2d)+sin(te)*(k2*p1)
```



```
""" Surface 3 : s1 = xep + k1*xe
                s2 = tep + k2*te + k0*ye """
def Psi1S3(u2,u2d,p1,te,p2,u1,k1):
    return k1*p1*cos(te) + sin(te)*(p1*u2d-u1*p2-p1*u2-p1*p2+p1*u2d)

def Psi2S3(p1,k0,te):
    return p1*k0*sin(te)

"""-----------------------------------------------------------------------------------------------------"""

vibes.beginDrawing()

"""Parametors"""
x1 = Interval(-70,70) #s1
x2 = Interval(-70,70) #s2

u1 = Interval(-1,1)
u2 = Interval(-1,1)
u2d = Interval(-1,1)
p1 = Interval(-0.1,0.1)
p2 = Interval(-0.1,0.1)
te = Interval(-np.pi,np.pi)

f1 = 1
f2 = 2

k0 = 0.6
k1 = 0.5
k2 = 0.8

nbcoup = 2
surface = 1 #Surface to work with

"""Bounding the function"""

t0 = time.time()
fsup1 = MaximIVIA(u2,u2d,p1,te,p2,u1,nbcoup,f1,surface,k0,k1,k2)
fmin1 = MinimIVIA(u2,u2d,p1,te,p2,u1,nbcoup,f1,surface,k0,k1,k2)
fsup2 = MaximIVIA(u2,u2d,p1,te,p2,u1,nbcoup,f2,surface,k0,k1,k2)
fmin2 = MinimIVIA(u2,u2d,p1,te,p2,u1,nbcoup,f2,surface,k0,k1,k2)
t1 = time.time()-t0
print("time to compute : ",t1)
print("bound of Psi1 : ",Interval(fmin1[0],fsup1[1]))
print("bound of Psi2 : ",Interval(fmin2[0],fsup2[1]))

"""V-stability"""
dt = 0.1
epsiSIVIA = 0.02
q = 0.9
p = 0.0001
psi1 = Interval(fmin1[0],fsup1[1])
psi2 = Interval(fmin2[0],fsup2[1])
vdot = Vdot(psi1[0],psi1[1],psi2[0],psi2[1],x1,x2,dt,epsiSIVIA,q,p)
Rver = Rverif(0,x1,x2,epsiSIVIA)
SepUni(x1,x2,epsiSIVIA)
```

Figure 21: Screenshot of the python algorithm