SOStab: a Matlab Toolbox for Transient Stability Analysis

Stéphane Drobot, Matteo Tacchi, Carmen Cardozo, Colin N. Jones

SOStab: a Matlab Toolbox for Transient Stability Analysis

Stéphane Drobot[1], Matteo Tacchi[2*], Carmen Cardozo[1] and Colin N. Jones[3]

October 11, 2023

Abstract

This paper presents a new Matlab toolbox, aimed at facilitating the use of polynomial optimization for stability analysis of nonlinear systems. Indeed, in the past decade several decisive contributions made it possible to recast this type of problems as convex optimization ones that are tractable in modest dimensions. However, available software requires their user to be fluent in Sum-of-Squares programming, preventing them from being more widely exploited by practitioners. To address this issue, SOStab entirely automates the writing and solving of optimization problems, and directly outputs relevant data for the user, while requiring minimal input. In particular, no specific knowledge of optimization is needed for implementation. The toolbox allows a user to obtain outer and inner approximates of the region of attraction of the operating point of different grid connected devices such as synchronous machines and power converters.

Keywords

AC power systems, software, transient stability analysis, region of attraction, sum-of-squares programming, toolbox, Matlab.

Acknowledgement

This work was supported by the Swiss National Science Foundation under the NCCR Automation project, grant agreement 51NF40_180545, and the French company RTE, under the RTE-EPFL partnership n°2022-0225.

[1]R&D division, Réseau de Transport d'Électricité, La Défense, France

[2]Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), GIPSA-lab, Grenoble, France.

[3]Autoamtic Control Laboratory, EPFL, Lausanne, Switzerland.

[*]Corresponding author. matteo.tacchi@gipsa-lab.fr. +33 4768 26235. 11 rue des Mathématiques, Grenoble Campus BP46, 38402 Saint-Martin-d'Hères Cedex, France.

1 Introduction

In the past decade, the potential application of Sum-of-Squares Programming (SoSP) to power systems stability assessment has been investigated [1, 2, 3, 4, 5]. In this line of research, the problem of transient stability analysis is reformulated as the search for a region $\mathcal{R}$ of the system state space, such that if the system state is initialized in $\mathcal{R}$, then the trajectories will exhibit stability properties (e.g. converging to a target –$\mathcal{R}$ is then called a *Region of Attraction* or *RoA*– or staying forever –i.e. as long as the system equations are valid– within a secure zone of the state space). Such a framework allows its user to take nonlinear behaviours into account, hence making it specifically suited for *transient* stability analysis. Indeed, transient stability is defined as the ability of the considered power system to come back to normal operation, after being subjected to a large perturbation that took it far from its equilibrium point [6], hence making linearization meaningless.

RoA on time horizons both infinite [7, 8, 9] and finite [10, 11] can then be computed by solving a SoSP problem. In many instances, a byproduct of this reformulation is the convexity of the resulting optimization problem: one is faced with Linear Matrix Inequalities (LMI), which corresponds to the framework known as Lasserre's Moment-SoS (for Sums-of-Squares) hierarchy and comes with convergence guarantees [12, 13, 14]. Nevertheless, approximation of finite horizon RoAs still suffers from various drawbacks: first of all, the computational complexity is polynomial in the dimension of the considered system (see [15] for details on that issue as well as [16, 17, 18] for existing solutions); second, many open questions remain about optimal conditioning of LMIs depending on various choices in the formulation of the problem; eventually, the issue that this article proposes to leverage, is the lack of a user-friendly interface for a practical implementation and application of the Moment-SoS hierarchy.

More precisely, the existing frameworks [19, 20, 21] require the user to write, code and solve SoS programming problems whose solutions describe the RoA approximation; in contrast, the SOStab toolbox entirely automates the SoS programming part of the framework and no knowledge on the Moment-SoS hierarchy is needed to run it. Instead, the SOStab toolbox requires minimal input (namely: dynamics, state constraints, equilibrium point, time horizon, target set and a complexity parameter d) and outputs the stability certificate that describes the RoA approximation, as well as plots of the RoA in chosen state coordinates.

The software is specifically designed to automate finite time horizon RoA approximation, with following interpretation in terms of transient stability: the finite time horizon is the time allowed for the system to return to normal operating conditions once the disturbance is cleared. The contribution of this work is twofold: a Matlab tollbox has been developed and made publicly available which allows users to compute RoA of non-linear dynamical systems. This paper, first demonstrates its capabilities with two illustrative test cases and then explains in deeper details the implemented procedure.

The article is organized as follows: first Section 2 recalls the mathematical definition and practical interpretation of finite time horizon RoA, and gives an overview of the SOStab toolbox. Basics about the SoSP framework are included in Appendix .1. Then, Section 3 introduces the models of the two case studies used in this work to illustrate the toolbox performance in Section 4. Section 5 provides more details on the developed tool and highlights the main contributions of this work before presenting the main conclusions of this work in Section 6.

2 Computing Regions of Attraction with SOStab

Consider an abstract differential system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)) \tag{1a}$$

with polynomial vector field $\boldsymbol{f} \in \mathbb{R}[\boldsymbol{x}]$ and equilibrium $\boldsymbol{x}^\star \in \mathbb{R}^n$. Define a security threshold $\Delta\boldsymbol{x} \in (0,\infty)^n$ and state constraint

$$\boldsymbol{x}(t) \in [\boldsymbol{x}^\star \pm \Delta\boldsymbol{x}], \tag{1b}$$

$[\boldsymbol{x}^\star \pm \Delta\boldsymbol{x}] := [x_1^\star - \Delta x_1, x_1^\star + \Delta x_1] \times \ldots \times [x_n^\star - \Delta x_n, x_n^\star + \Delta x_n]$.

Definition 1. Given a time horizon $T > 0$ and a closed target set $\mathbb{M} \subset \mathbb{X}$, the Region of Attraction $\mathcal{R}_T^{\mathbb{M}}$ of $\mathbb{M}$ in time T is defined as

$$\mathcal{R}_T^{\mathbb{M}} := \left\{ \boldsymbol{x}(0) \in \mathbb{R}^n \ : \ \begin{array}{c} \forall t \in [0,T], \ (1) \text{ holds} \\ \boldsymbol{x}(T) \in \mathbb{M} \end{array} \right\}. \tag{2}$$

SOStab is specifically designed to compute such an $\mathcal{R}_T^{\mathbb{M}}$ for ellipsoids $\mathbb{M}$ described by

$$\mathbb{M} := \{\boldsymbol{x} \in \mathbb{R}^n \ : \ \|\mathbf{A}(\boldsymbol{x} - \boldsymbol{x}^\star)\| \leq \varepsilon\} \tag{3}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a reshaping matrix such that $\det(\mathbf{A}) = 1$, and $\varepsilon > 0$ is an error tolerance.

From a computational viewpoint, considering finite time horizon RoA improves the properties of the resulting SoS programs: they are convex, while their infinite time horizon counterparts are bilinear (and thus nonconvex), which has important consequences on the convergence properties of the corresponding algorithms (see e.g. [3]). In practice, this is a reasonable assumption as operational points of physical system, such as power systems, are constantly moving, making the results of the analysis relevant for a limited time window. In exchange for the finite time horizon, it is necessary to consider target sets that are not reduced to a point, defined by parameters $\mathbf{A}$ and ε; in SOStab, the default value for $\mathbf{A}$ is the identity matrix, but sometimes (e.g. when variables evolve on different time scales), it is useful to make a different choice. Eventually, we embed the whole problem into a secure region $[\boldsymbol{x}_{eq} \pm \Delta\boldsymbol{x}]$, to take into account the physical limits of the system: this is the upper and lower bounds of state variables.

SOStab takes as input the problem data $(\boldsymbol{f}, \Delta\boldsymbol{x}, T, \mathbf{A}, \varepsilon)$, as well as a modelling parameter $d \in 2\mathbb{N}$, and outputs the following objects for outer and inner RoA estimation:

$$\boldsymbol{y}_d^{out} = \begin{pmatrix} \lambda_d^{out} & v_d^{out} & w_d^{out} \end{pmatrix} \tag{4a}$$

$$\boldsymbol{y}_d^{in} = \begin{pmatrix} \lambda_d^{in} & v_d^{in} & w_d^{in} \end{pmatrix}, \tag{4b}$$

with $\lambda_d^{in/out} \geq 0$, $v_d^{in/out} \in \mathbb{R}_d[t,\boldsymbol{x}]$ degree d polynomials in $(t,\boldsymbol{x})$ and $w_d^{in/out} \in \mathbb{R}_d[\boldsymbol{x}]$ degree d polynomials in $\boldsymbol{x}$, such that the following inclusion guarantees hold:

$$\mathcal{R}_T^{\mathbb{M}} \subset \{\boldsymbol{x} \in \mathbb{R}^n \ : \ v_d^{out}(0,\boldsymbol{x}) \geq 0\} =: \mathcal{R}_d^{out} \tag{5a}$$

$$\mathcal{R}_T^{\mathbb{M}} \supset \{\boldsymbol{x} \in \mathbb{R}^n \ : \ v_d^{in}(0,\boldsymbol{x}) < 0\} =: \mathcal{R}_d^{in}. \tag{5b}$$

In words, v_d^{out} (resp. v_d^{in}) defines an outer (resp. inner) approximation $\mathcal{R}_d^{out}$ (resp. $\mathcal{R}_d^{in}$) of $\mathcal{R}_T^{\mathbb{X}}$. Moreover, the following relations hold between outputs:

$$\lambda_d^{in/out} = \int w_d^{in/out}(\boldsymbol{x}) \, d\boldsymbol{x} \tag{6a}$$

$$w_d^{in/out}(\boldsymbol{x}) \geq \max\Big(0, v_d^{in/out}(0,\boldsymbol{x}) + 1\Big). \tag{6b}$$

As a result, $\lambda_d^{in/out}$ give relevant information on the size of the computed RoA estimates: the smaller they are, the more accurate the approximation. $w_d^{in/out}$ is often used to check the behavior of the numerical solver: if the coefficients of $w_d^{in/out} - 1$ are too small, then $w_d^{in/out} \simeq 0$, and the solver failed to detect the RoA. Eventually, the framework comes with precision guarantees:

$$\lambda_d^{out} \underset{d\to\infty}{\searrow} \mathrm{vol}\left(\mathcal{R}_T^{\mathbb{M}}\right) \tag{7a}$$

$$\lambda_d^{in} \underset{d\to\infty}{\searrow} \mathrm{vol}\left([\boldsymbol{x}^\star \pm \Delta \boldsymbol{x}] \setminus \mathcal{R}_T^{\mathbb{M}}\right) \tag{7b}$$

$$\mathrm{vol}\left(\mathcal{R}_d^{in/out}\right) \underset{d\to\infty}{\longrightarrow} \mathrm{vol}\left(\mathcal{R}_T^{\mathbb{M}}\right) \tag{7c}$$

where vol denotes the n-dimensional volume of a set. In words, when the modelling parameter d goes to infinity, the volume of the error made by the inner and outer approximation schemes converges to 0 (see Appendix .1 or [10, 11] for further details).

3 Models for case study

In this section we present the model of two different test cases, proposed in the literature, to illustrate the tool capabilities, flexibility and performance. The first one is the well known synchronisation loop of classic grid-following converters: the phase locked loop (PLL [22, 23, 24]). The second in a multi-machine system considering reduced order models (ROM) of synchronous generators.

3.1 Phase Locked Loop 2$^{\text{nd}}$ order model

Figure 1 shows a generic PLL block diagram [22]. Here an angular state variable θ is required to match a reference θ_{ref}; to that end, the system computes the sine of the phase difference ϕ, multiplies it by some gain K and takes it through a low-pass filter with transfer function $F(s) = \frac{1+\tau_2 s}{\tau_1 s}$, resulting in the following differential system:

$$\begin{pmatrix} \dot{\phi} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \omega \\ -K\frac{\tau_2}{\tau_1}\cos(\phi)\omega - \frac{K}{\tau_1}\sin(\phi) \end{pmatrix}. \tag{8}$$

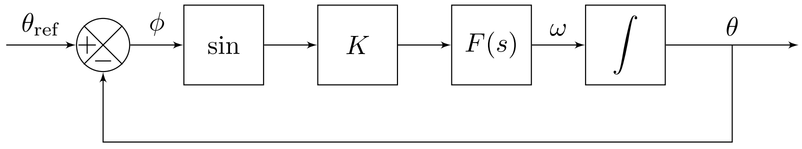

Figure 1: Block scheme of a PLL system.

Following [22], in the PLL setting the time constants τ_1 and τ_2 are functions of the gain K, a natural frequency ω_n and a damping ratio ζ:

$$\tau_1 = \frac{K}{\omega_n^2} \qquad \tau_2 = \frac{2\zeta}{\omega_n} \tag{9}$$

3.2 Interconnected synchronous machines 2nd order model

The second test case considered for SOStab consists of three synchronous machines, each represented by a voltage with constant magnitude V_i and time varying phase angle ϕ_i. The corresponding frequencies are ruled by the swing equation:

$$\ddot{\phi}_i = \dot{\omega}_i = -\lambda_i \omega_i + \frac{1}{M_i}\left(P_i^m - P_i^e(\boldsymbol{\phi})\right) \tag{10}$$

where P_i^e is the electrical power output of machine i (and P_i^m its mechanical power input), M_i is the inertia constant of machine i and λ_i is related to its damping coefficient. The machines are linked to one another through pure reactances X_{ij} according to Figure 2, resulting in the following lossless power flow equation [1, 4]:

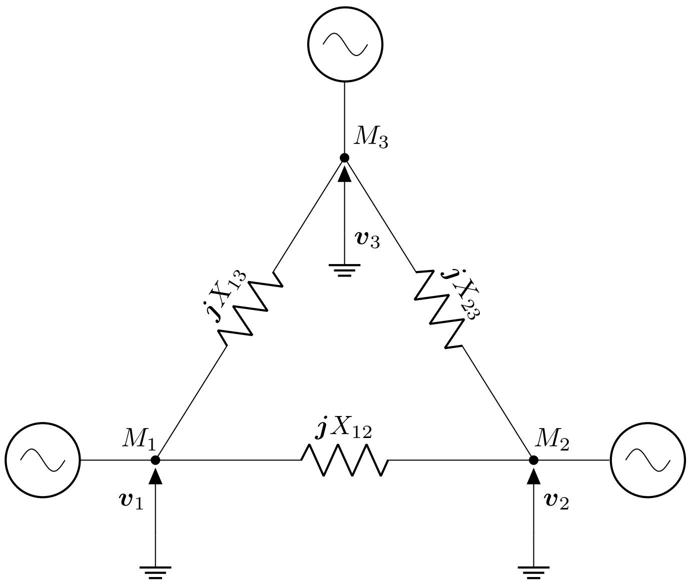

Figure 2: Three synchronous machines connected in a cycle.

$$P_i^e(\boldsymbol{\phi}) = \sum_{j \neq i} \frac{V_i V_j}{X_{ij}} \sin\left(\phi_i - \phi_j\right). \tag{11}$$

4 Case study

4.1 Installation of the toolbox

SOStab is a freeware subject to the General Public Licence (GPL) policy available for Matlab. It can be downloaded at:

`https://github.com/droste89/SOStab`

SOStab requires YALMIP [19], as well as a semidefinite solver. Mosek [25] is used by default, but it can be replaced by any other solver, provided they are installed and interfaced through YALMIP.

4.2 RoA of the Phase Locked Loop system

This section presents the implementation of SOStab to compute the RoA of the PLL system (8), where sine and cosine have been replaced by their degree 10 Taylor expansions (denoted `si` and `co` respectively), so that the vector field $\boldsymbol{f}$ is polynomial. First, parameters are assigned numerical values:

```
K = 1;            phib = pi;
omegan = 10.813;  omegab = 20*pi;
zeta = 1.3303;    Deltax = [phib; omegab]
```

Obviously, considering (8), one can compute the following equilibrium:

```
phieq = 0; omegaeq = 0;
xeq = [phieq ; omegaeq];
```

Then, an instance of the SOStab problem is created with the line of code:

```
PLL = SOStab(xeq, Deltax);
```

and the dynamics of the system are introduced:

```
tau1 = K/(omegan^2);  tau2 = 2*zeta/omegan;
PLL.dynamics = [PLL.x(2); ...
-K/tau1*(si(PLL.x(1)) + tau2*co(PLL.x(2))];
```

where `PLL.x` denotes the variable $\boldsymbol{x}$ of the system (automatically defined by `SOStab` when it is first called).

Now that the problem is defined, an outer approximation of the time $T = 1$ RoA of $\mathbb{M} = \left\{(\phi, \omega) \in \mathbb{R}^2 \;:\; 20\phi^2 + 0.05\omega^2 \leq 1.7^2\right\}$ is computed using the following command:

```
T=1;        epsilon = 1.7;          d=16;
A = [[20^(1/2) 0] ; [0 20^(-1/2)]];
[vol, vc, wc] = PLL.SoS_out(d,T,epsilon,A);
```

It returns the approximate surface `vol`$= \lambda_d^{out}$ of the computed RoA estimate in the phase space, as well as two vectors `vc` and `wc` consisting of the coefficients of polynomials v_d^{out} and w_d^{out} respectively. We run SOStab for the PLL system for various values of d and compile the results in Table 1. As λ_d^{out} is a proxy for the size of the outer RoA estimate, the smaller it is, the more accurate the approximation. Here one can observe that as expected the accuracy increases wirh d at the price of higher computational time.

Once the optimisation problem is solved, graphical representation in two dimension are also available using:

d	λ_d^{out}	CPU time (s)
4	4.0000	2.8169
8	3.5892	4.6729
12	3.1284	14.5575
16	2.9346	45.2185

Table 1: Outputs of SOStab depending on precision parameter d

```
  PLL.plot_roa(1,2,'outer');
```

where the first two arguments indicate the indices of the represented variables (respectively in abscissa and ordinate), in case there are more than two. The string `'outer'` indicates that the toolbox plots an outer estimate of the RoA.

For inner RoA approximation [11], the commands are

```
[vol, vc, wc] = PLL.SoS_in(d, T, epsilon);
VdP.plot_roa(1, 2, 'inner',1);
```

Here the last argument with value `1` is an optional argument that asks SOStab to also represent the target set in the figure. This yields the plot represented in Figure 3, which can be compared to [22, Fig. 10, Right].

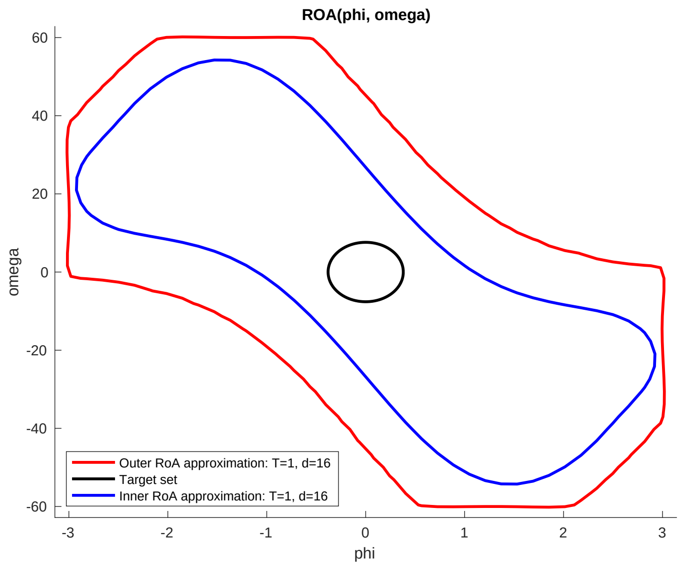

Figure 3: Inner and outer RoA approximations for the PLL system.

It can also be interesting to display 3d-plots of polynomials v_d^{out} and w_d^{out}, which can be performed by the commands (see Figure 4: one retrieves the shape of the inner RoA estimate):

```
PLL.plot_v(1, 2, 'outer');
PLL.plot_w(1, 2, 'outer');
```

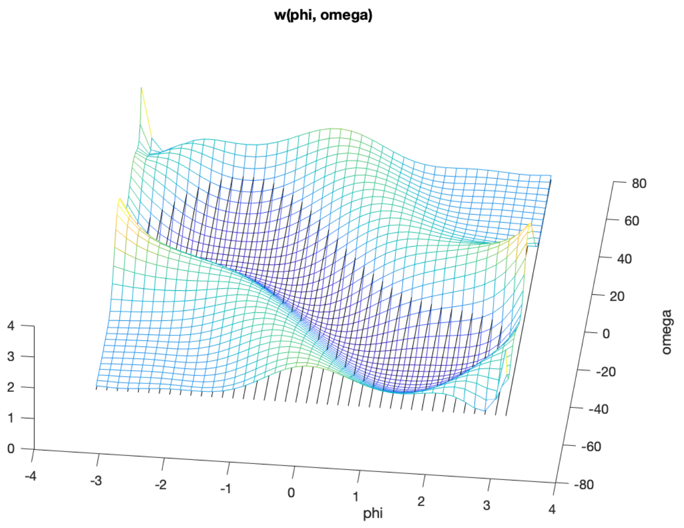

Figure 4: Plot of w_d^{in} for the PLL system

Of course, one can also represent the certificates v_d^{in} and w_d^{in} obtained in inner approximation, simply by setting the last argument at `'inner'`.

4.3 RoA of the multi-machines system

Instead of performing Taylor expansions as in the previous section, it is also possible to directly tackle trigonometric functions, through an algebraic change of variables [4]. This approach is also supported by SOStab, which we demonstrate on the 2nd order model of three interconnected synchronous machines (10)–(11). By choosing a phase reference aligned with θ_3, one sets $\theta_3 = 0$ rad and $\omega_3 = 0$ rad/s. Moreover, the parameters are fixed according to Table 2:

Parameter	λ_1	λ_2	M_i	P_1^m	P_2^m
Value	0.4 s^{-1}	0.5 s^{-1}	20 pu	0 pu	1 pu
Parameter	V_i	X_{12}	X_{23}	X_{13}	
Value	1 pu	0.1 pu	0.1 pu	0.05 pu	

Table 2: Parameters for interconnected synchronous machines.

This way, one finds back the equations studied in [1, 4]:

$$\dot\theta_1 = \omega_1, \qquad \dot\theta_2 = \omega_2 \tag{12a}$$
$$\dot\omega_1 = -\sin\theta_1 - 0.5\sin(\theta_1-\theta_2) - 0.4\,\omega_1 \tag{12b}$$
$$\dot\omega_2 = -0.5\sin\theta_2 - 0.5\sin(\theta_2-\theta_1) - 0.5\,\omega_2 + 0.05 \tag{12c}$$

with state variable $\boldsymbol{\sigma} = (\theta_1, \theta_2, \omega_1, \omega_2) \in \mathbb{R}^4$, stable equilibrium given by $\boldsymbol{\sigma}^\star = (0.02, 0.06, 0, 0)$ and non-polynomial dynamics. Here, a preprocessing $\boldsymbol{x} := \boldsymbol{\varphi}(\boldsymbol{\sigma})$ is required from the user to recast the dynamics under a polynomial form; each phase variable θ_i is represented by its sine and cosine, through the change of variables:

$$\boldsymbol{x} = (\sin\theta_1, \cos\theta_1, \sin\theta_2, \cos\theta_2, \omega_1, \omega_2) \in \mathbb{R}^6 \tag{13}$$

Then, one can prove (similarly to e.g. [1, 4]) that the new variable $\boldsymbol{x}$ has polynomial dynamics: $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$, with $\boldsymbol{f} \in \mathbb{R}[\mathrm{x}]^6$, and equilibrium $\boldsymbol{x}^\star = \boldsymbol{\varphi}(\boldsymbol{\sigma}^\star)$. Then, the RoA approximation problem is defined with

```
Sync = SOStab([sin(0.02);cos(0.02);
sin(0.06); cos(0.06);0;0], [1;1;1;1;pi;pi],
[1,2;3,4]);
```

Here the first two arguments are identical as in the polynomial setting: they indicate the equilibrium point and admissible deviation for the variable with polynomial dynamics. The third argument is specific to systems with trigonometric dynamics, and identifies sines and cosines of phase variables θ_i in the recasted $\boldsymbol{x}$: the line `[1,2]` (resp. `[3,4]`) means that $\begin{pmatrix}x_1 & x_2\end{pmatrix} = \begin{pmatrix}\sin\theta_1 & \cos\theta_1\end{pmatrix}$ (resp. $\begin{pmatrix}x_3 & x_4\end{pmatrix} = \begin{pmatrix}\sin\theta_2 & \cos\theta_2\end{pmatrix}$). Then, `Sync.dynamics`, `Sync.SoS_outer` and `Sync.SoS_inner` are the same as in the polynomial setting (but one should be careful to input the dynamics of $\boldsymbol{x}$ and not of $\boldsymbol{\sigma}$). However, to plot the result in the original phase variables, the code is slightly modified in its first argument, as the user needs to explicitly identify which of the original variables in $\boldsymbol{\sigma}$ should be represented:

```
Sync.plot_roa([1,2],[3,4],'outer', 1);
```

Here, `[1,2]` (resp. `[3,4]`) means that the plot abscissa (resp. ordinate) represents the variable θ_1 (resp. θ_2) such that $\begin{pmatrix}\sin\theta_1 & \cos\theta_1\end{pmatrix} = \begin{pmatrix}x_1 & x_2\end{pmatrix}$ (resp. $\begin{pmatrix}\sin\theta_2 & \cos\theta_2\end{pmatrix} = \begin{pmatrix}x_3 & x_4\end{pmatrix}$). This yields, in the case of our power system, the plot of Figure 5, which reproduces (at a lower degree) the result of [4, Figure 2].

Here we only display the outer approximation: at degree $d = 6$, the precision is not enough to obtain a relevant inner RoA approximation, and the computed $\mathcal{R}_d^{in}$ is empty. With four variables (among which two angular phases, leading to six recast variables), going higher in the degree would require a computing power not available on a standard laptop.

Remark 1. One could also represent the RoA in a phase space (e.g. in the (θ_1, ω_1) plane) with the following command, which mixes a pair of indices (for θ_1) and a single index (for ω_1):

```
Sync.plot_roa([1,2],5,'outer', 1);
```

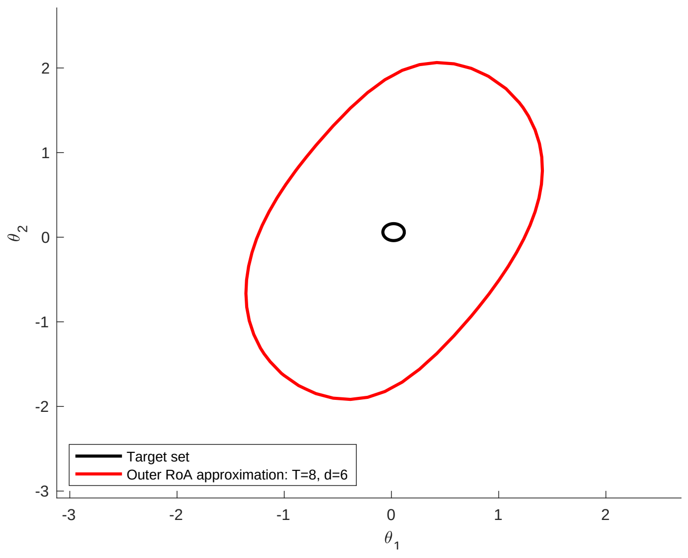

Figure 5: Outer RoA approximation for the power system.

5 Workings of the toolbox

5.1 Overview of the toolbox

The problem formulated by the user and given to the toolbox must have a specific form. First of all, it must be already in polynomial form, which means that only products of variables are allowed. The toolbox requires the following minimal input:

$$\begin{aligned} &\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}), \quad \boldsymbol{f} \in \mathbb{R}[\mathbf{x}]^n, \quad T > 0 \\ &\boldsymbol{x}^\star \in \{\boldsymbol{x} \in \mathbb{R}^n \ : \ \boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}\} \\ &\Delta\boldsymbol{x} \in (0,\infty)^n, \quad d \in 2\mathbb{N}, \quad \varepsilon > 0. \end{aligned}$$

Then, it identifies the problem to be solved as: "compute approximations $\mathcal{R}_d$ of $\mathcal{R}_T^{\mathbb{M}}$ with dynamics $\boldsymbol{f}$, admissible set $[\boldsymbol{x}^\star \pm \Delta\boldsymbol{x}]$ and target set $\mathbb{M} = \{\boldsymbol{x} \in \mathbb{R}^n \ : \ \|\boldsymbol{x} - \boldsymbol{x}^\star\| \leq \epsilon\}$ ".

SOStab also admits two optional input arguments:

- a shape matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ such that $\det(\mathbf{A}) = 1$ reshapes the target set as $\mathbb{M} = \{\boldsymbol{x} \in \mathbb{R}^n \ : \ \|\mathbf{A}(\boldsymbol{x} - \boldsymbol{x}^\star)\| \leq \epsilon\}$;
- in case the system at hand involves trigonometric functions of phase variables $\theta_1, \ldots, \theta_N$, it is also possible to specify a phase index matrix $\mathbf{Z} \in \mathbb{R}^{N\times 2}$ whose first (resp. second) column consists of the indices of the sines (resp. cosines) of the θ_i in the recasted variable $\boldsymbol{x} = \boldsymbol{\varphi}(\boldsymbol{\sigma})$.

The use of the toolbox consists of four steps (see Figure 6):

1. The initialization is performed from input $(\boldsymbol{x}^\star, \Delta\boldsymbol{x})$ (and optional input $\mathbf{Z}$); it defines the admissible set $[\boldsymbol{x}^\star \pm \Delta\boldsymbol{x}]$ and identifies the dimension and variables of the system.

2. The user the inputs the dynamics $\boldsymbol{f}$ of the system and adjusts optional settings of the toolbox.
3. The RoA approximation itself is performed from input (d, T, ε) (with optional input $\mathbf{A}$); it outputs the SDP solutions $\boldsymbol{y}_d^{in/out} = (\lambda_d^{in/out}, v_d^{in/out}, w_d^{in/out})$.
4. Graphic representations of the solutions can eventually be plotted; the choice of the plots abscissa and ordinate and plotting options is up to the user.

5.2 Initialization

SOStab is initialized through the following command:

```
RoA_pb = SOStab(x_eq, delta_x); % or
RoA_pb = SOStab(x_eq, delta_x, angle_ind);
```

where `x_eq`= $\boldsymbol{x}^\star$, `delta_x`= $\Delta\boldsymbol{x}$ and `angle_ind`= $\mathbf{Z}$.

Remark 2. Note that the Moment-SoS hierarchy framework can be used with admissible sets different from $[\boldsymbol{x}^\star \pm \Delta\boldsymbol{x}]$ (e.g. ellipsoids, possibly independent from the equilibrium). SOStab focuses on boxes to match the intuition of physical limits on a system. However, in some cases, such description induces bad conditioning in the considered LMIs. Future versions of SOStab will include a broader range of admissible sets.

The initial call creates an instance of the class, and defines a number of internal properties, among which one can find the following useful ones:

- internal copies of the inputs `x_eq` and `delta_x` (and optionally `angle_ind`, empty by default)
- `dimension`: problem dimension (number of variables)
- `x`: a YALMIP sdpvar polynomial object, of the dimension of the problem. It represents the variable $\boldsymbol{x}$ and is called by the user to define the dynamics of the system
- `t`: sdpvar polynomial of size 1, representing the time variable t, which can be needed to define the dynamics of the system (if non-autonomous)
- `solver`, the choice of the solver used in the optimization, defined as Mosek by default, it can also be SeDuMi
- `verbose`, the value of the verbose parameters of the YALMIP optimization calls, defined at 2 by default (all numerical SDP solver info displayed), it can also be 1 (selection of info displayed) or 0 (no info about the numerical resolution of the underlying SDP)
- `dynamics`, a YALMIP polynomial defining the polynomial dynamics $\boldsymbol{f}$ of the system.

The dynamics of the system are added into the class by the user by allocating a value to `RoA_pb.dynamics`. The dynamics must be a vector of size `dimension`, each row corresponding to the dynamics of one variable. To define dynamics, one will use the sdpvar object `x` created inside the class. For instance, defining $(\dot{x_1}, \dot{x_2}) = (-x_1, -x_2^3)$ would be:

```
RoA_pb.dynamics = [-RoA_pb.x(1);
                   -RoA_pb.x(2)^3];
```

Remark 3. If the system at hand involves trigonometric functions of phase variables, SOStab defines the additional property `angle_eq`, which stands for the vector $\boldsymbol{\theta}^\star$ – the equilibrium angles – recalculated from the values of their sine and cosine (empty if no phase variables are involved)

5.3 RoA estimation

The inner and outer RoA estimation of the system defined by the call of `SOStab` are then computed by the methods `SoS_in` and `SoS_out`, respectively. Additionnal properties are related to a specific solution of the optimization problem. They are calculated at each call of the optimization and stored until the next call, *i.e.* each of them correponds to the previous optimization call:

- internal copies of the inputs `d`, `T`, `epsilon` (and optional `A`, set to identity by default); recall that `d` is the degree of the polynomials $v_d^{out/in}$ and $w_d^{out/in}$
- `vcoef_outer`, the coefficients of the solution v_d^{out} for the last calculated outer approximation of the ROA
- `wcoef_outer`, the coefficients of the solution w_d^{out} for the last calculated outer approximation of the ROA
- `vcoef_inner`, the coefficients of the solution v_d^{in} for the last calculated inner approximation of the ROA
- `wcoef_inner`, the coefficients of the solution w_d^{in} for the last calculated inner approximation of the ROA
- `solution`, a volume approximation of the last calculated ROA, *ie* the solution $\lambda_d^{out/in}$ of the optimization problem

Storing the input variables $(d, T, \varepsilon, \mathbf{A})$ allows the toolbox to reuse them at the plotting step, which is performed as follows:

`plot_roa` takes as inputs the two indices i, j of the variables on which to project the RoA, a string to choose between `'inner'` and `'outer'` approximation, and four optional arguments: an `int` (1 or 0) – 0 by default – to choose to plot the target or not, two strings indicating the axes names – x_i and x_j by default – and the size of the plotting mesh – $(40, 40)$ by default. It plots the expected slice of the ROA, with all other variables at equilibrium. If both inner and outer approximations are called sequentially for the same variables, the two plots will appear on the same Figure.

`plot_v` and `plot_w` take the same inputs as `plot_roa`. They respectively plot the graph of v_d and w_d in 3D (with non-selected variables at equilibrium).

These four steps of SOStab (initialization, dynamics setting, numerical RoA estimation and graphic representations) are summarized in Figure 6.

5.4 Added value of SOStab

As detailed in Appendix .1 and [10, 11], the Moment-SoS hierarchy consists in solving SoS programming problems of increasing size, which requires to follow a number of steps, related to real algebraic geometry and optimization:

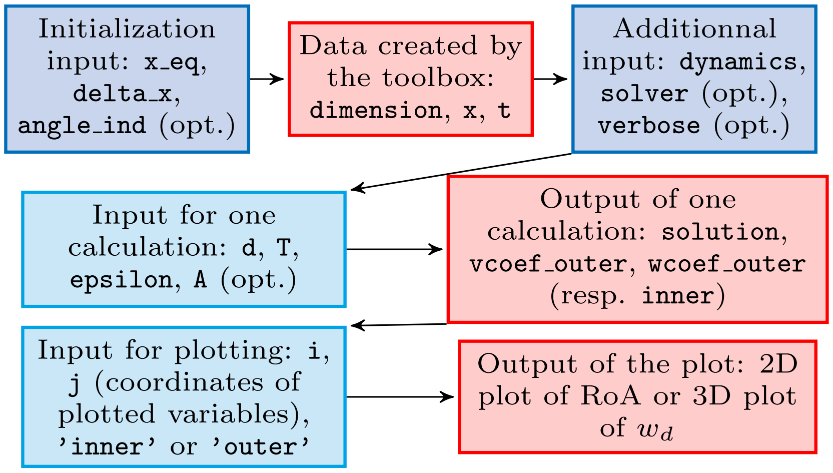
Initialization input: x_eq, delta_x, angle_ind (opt.)
Data created by the toolbox: dimension, x, t
Additionnal input: dynamics, solver (opt.), verbose (opt.)
Input for one calculation: d, T, epsilon, A (opt.)
Output of one calculation: solution, vcoef_outer, wcoef_outer (resp. inner)
Input for plotting: i, j (coordinates of plotted variables), 'inner' or 'outer'
Output of the plot: 2D plot of RoA or 3D plot of w_d

Figure 6: Flowchart of the toolbox workings

1. Defining the geometric characteristics of the problem (polynomial dynamics $\boldsymbol{f}$, time horizon T, admissible and target sets $\mathbb{X}$ and $\mathbb{M}$)

2. Defining an algebraic description of these geometric characteristics: polynomials $\boldsymbol{g} \in \mathbb{R}[\mathrm{x}]^m$ s.t. $\mathbb{X} = \{\boldsymbol{x} \,:\, \boldsymbol{g}(\boldsymbol{x}) \geq 0\}$ and $\boldsymbol{h} \in \mathbb{R}[\mathrm{x}]^\ell$ s.t. $\mathbb{M} = \{\boldsymbol{x} \,:\, \boldsymbol{h}(\boldsymbol{x}) \geq 0\}$

3. Coding a method to integrate polynomials over $\mathbb{X}$ (i.e. computing the moments of the Lebesgue measure on $\mathbb{X}$)

4. Writing the SoS programming problems with explicit positivity constraints; this requires introducing internal SoS certificates as decision variables

5. Recasting SoS constraints as LMIs, solving the resulting SDP problem and converting the solution back to the polynomial framework

6. Extracting the corresponding certificates $v_d^{in/out}, w_d^{in/out}$ and using them to characterize and plot relevant representations of the computed RoA approximation.

Existing frameworks [19, 20, 21] have been designed to automate step 5 which appears in all instances of SoS programming. As a result, they are very flexible in their use, but they also require their user to perform steps 1–4 and 6 by hand, which requires solid knowledge in SoS programming and may be time consuming and prone to human errors (especially in step 4) ; hence their use is usually not smooth, even for experts. In contrast, with SOStab, only step 1 is left to the user, and all the other operations are automatically performed.

More precisely, intrinsic properties of the dynamical system are defined as presented in section 5.2, and then settings for finite horizon RoA are the input of methods `SoS_in` and `SoS_out`; with this, steps 2–5 are performed through a call to YALMIP, for inner and outer RoA approximation respectively, and output an optimal value `solution` and the coefficients of the optimal polynomials $v_d^{in/out}$ and $w_d^{in/out}$. Currently, the inner approximation has limitations on some of the instances tested, due to technical specificities of the corresponding SoS programming problem, which can cause ill conditioning of the corresponding SDP.

Remark 4. In the current version of the toolbox, when using an optional argument, one should also specify all optional arguments that appear before in the method call.

6 Conclusion and future works

This work presented a new Matlab Toolbox called SOStab, which aims at helping a non-expert in polynomial optimization to use the frameworks developed in [10, 11, 4] through a plug-and-play interface. Taking only ODE-related inputs (such as dynamics, admissible and target sets, time horizon) as well as an even complexity parameter $d \in 2\mathbb{N}$, the toolbox fully automates writing, recasting and solving the corresponding SoS programming problem, and outputs stability certificates v_d and w_d which can be evaluated at a given initial condition to assess its stability. This is in sharp contrast with existing frameworks [19, 20, 21] which require the users to take upon themselves the burden of properly designing and coding the SoS programming problems that correspond to their RoA approximation problem. The benefits of this contribution are the following:

(a) Knowledge on SoS programming becomes optional to use the Lasserre hierarchy for RoA approximation.

(b) The user input is significantly reduced, limiting implementation efforts.

(c) The toolbox comes with a plug-and-play design that allows one to repeat multiple experiments, reproduce existing results from the literature and solve new problems.

However, some limitations remain to be leveraged. For instance, while convex, SDP problems can be ill-conditioned, which sometimes results in poor numerical behavior with $w_d^\star \simeq 1$ and meaningless plots. It is possible to rescale SoS constraints to mitigate that phenomenon, although finding the appropriate rescalings is non-trivial. Also, inner RoA approximation requires an algebraic representation of the boundary $\partial\mathbb{X}$ of the admissible set $\mathbb{X}$ [11], and while choosing a box is the most physically relevant (and the easiest to integrate polynomials on), it induces some numerical difficulties that would not arise if $\mathbb{X}$ were described by a single polynomial. This can be solved either by improving the inner approximation scheme on a box, or by changing the admissible set $\mathbb{X}$.

Another issue related to the numerical convergence of Lasserre's hierarchy, is that the canonical basis of monomials is far from being the best choice (see the last concluding statement in [10]) and a way to work with other bases of polynomials (especially orthonormal bases) would help improving all existing results.

Last but not least, at a given number n of variables (such that $\boldsymbol{x} \in \mathbb{R}^n$ and given precision degree d, SOStab (and the Lasserre hierarchy in general) requires to solve size $\binom{n+d+1}{d}$ SDP problems, which quickly becomes intractable on any computer. To tackle this issue, structure exploiting methods have been developed in [15, 17], which consist in splitting the underlying SDP problems into problems of smaller size, while keeping as much computing precision as possible; these techniques bear the potential for scaling the hierarchy up to more realistic power systems such as fully modelled power converters or low order models of distributions network (which exhibit a radial structure one can exploit in computations) Future works on the SOStab class will include:

(a) Running the toolbox on more sophisticated case studies such as power converters

(b) Improving the inner approximation scheme to increase the accuracy of each relaxation

(c) Supporting richer admissible $\mathbb{X}$ and target $\mathbb{M}$ sets, such as ellipsoids, ℓ^p-balls, annuli...

(d) Extending the toolbox to other frameworks e.g. [8, 9]

(e) Supporting bases of polynomials other than monomials (Chebyshev, Legendre, trigonometric polynomials)

(f) Exporting the toolbox to other softwares compatible with existing SDP solvers, such as Julia or Python

(g) Adding structure exploiting methods to scale the method to higher dimensional dynamical systems

References

[1] M. Anghel, F. Milano and A. Papachristodoulou, "Algorithmic construction of Lyapunov functions for power system stability analysis", *IEEE Transactions on Circuits and Systems I* 60(9):2533-2546, 2013.

[2] M. Tacchi, B. Marinescu, M. Anghel, S. Kundu, S. Benahmed and C. Cardozo, "Power system transient stability analysis using sum of squares programming", in *Power Systems Computation Conference*, 2018.

[3] S. Izumi, H. Somekawa, X. Xin and T. Yamasaki, "Estimating regions of attraction of power systems by using sum of squares programming", *Electrical Engineering* 100:2205–2216, 2018.

[4] C. Josz et al."Transient stability analysis of power systems via occupation measures", in *2019 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2019.

[5] A. Oustry et al. "Maximal positively invariant set determination for transient stability assessment in power systems", *IEEE 58th Conference on Decision and Control (CDC)*, Nice (France), pp. 6572–6577, 2019.

[6] N. Hatziargyriou et al. "Definition and classification of power system stability – revisited & extended", *IEEE Transactions on Power Systems* 36(4):3271–3281, 2021.

[7] Z.W. Jarvis-Wloszek, *Lyapunov based analysis and controller synthesis for polynomial systems using sum-of-square optimization.* PhD thesis, University of California, 2003.

[8] M. Korda, D. Henrion and C.N. Jones, "Convex computation of the maximal positively invariant set for polynomial control systems", *SIAM Journal on Control and Optimization* 52(5):2944–2969, 2014.

[9] A. Oustry, M. Tacchi and D. Henrion, "Inner approximations of the maximal positively invarian set for polynomial dynamical systems", *IEEE Control Systems Letters* 3(3):733–738, 2019.

[10] D. Henrion and M. Korda, "Convex computation of the region of attraction of polynomial control systems", *IEEE Transactions on Automatic Control* 59(2):297–312, 2013.

[11] M. Korda, D. Henrion and C.N. Jones, "Inner approximations of the region of attraction for polynomial dynamical systems", *IEEE Transactions on Automatic Control* 59(2):297–312, 2013.

[12] J.B. Lasserre, *Moments, Positive Polynomials and Their Applications.* Imperial College Press, 2010.

[13] D. Henrion, M. Korda and J.B. Lasserre, *The Moment-SOS Hierarchy.* World Scientific, 2021.

[14] M. Tacchi, "Convergence of Lasserre's hierarchy: the general case", *Optimization Letters* 16(3):1015–1033, 2022.

[15] M. Tacchi, *Moment-SOS hierarchy for large scale set approximation. Application to power systems transient stability analysis.* PhD thesis, INSA Toulouse, 2021.

[16] M. Tacchi, C. Cardozo, D. Henrion, J.B. Lasserre, "Approximating regions of attraction of a sparse polynomial differential system", *IFAC-PapersOnLine*, 53(2):3266–3271, 2020.

[17] I. Subotić et al. "A Lyapunov framework for nested dynamical systems on multiple time scales with application to converter-based power systems", *IEEE Transactions on Automatic Control*, 66(12):5909–5924, 2021.

[18] J. Wang et al. Exploiting term sparsity in moment-SoS hierarchy for dynamical systems. *ArXiv preprint* 2111.08347, 2021.

[19] J. Löfberg, "Pre- and post-processing sum-of-squares programs in practice", *IEEE Transactions on Automatic Control* 54(5):1007–1011, 2009.

[20] D. Henrion, J.B. Lasserre and J. Löfberg, "GloptiPoly 3: moments, optimization and semidefinit programming", *Optimization Methods and Software* 24(4-5):761–779, 2009.

[21] A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, P.A. Parrilo, M.M. Peet and D. Jagt, *SOSTOOLS – Sum of Squares Optimization Toolbox for MATLAB.*

[22] T.-C. Wang, S. Lall and T.-Y. Chiou, "Polynomial method for PLL controller optimization", *Sensors* 11:6575–6592, 2011.

[23] C. Zhang, M. Molinas, J. Lyu, H. Zong and X. Cai, "Understanding the nonlinear behaviour and synchronizing stability of a grid-tied VSC under grid voltage sags", *IEEE 8th Renewable Power Generation Conference (RPG)*, Shanghai (China), 2019.

[24] C. Zhang, M. Molinas, Z. Li and X. Cai, "Synchronizing stability analysis and region of attraction estimation of greed-feeding VSCs using Sum-of-Squares programming", *Frontiers in Energy Research* 8, article 56, 2020.

[25] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 10.0*, 2022.

.1 Lasserre hierarchy for Region of attraction

In this section, the generic problem of computing the finite time RoA of a given target set is presented, along with the SoS framework to address it. Consider the system

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{14a}$$

with vector field $\boldsymbol{f} \in C^\infty(\mathbb{R}^n)^n$ and state constraint

$$\boldsymbol{x}(t) \in \mathbb{X} \tag{14b}$$

for some subset $\mathbb{X} \subset \mathbb{R}^n$ representing security constraints.

Definition 2. Given a time horizon $T \in (0, \infty]$ and a closed target set $\mathbb{M} \subset \mathbb{X}$, the Region of Attraction (RoA) of $\mathbb{M}$ in time T is defined as

$$\mathcal{R}_T^{\mathbb{M}} := \left\{ \boldsymbol{x}(0) \in \mathbb{R}^n \ : \ \begin{array}{c} \forall t \in [0,T), \quad \boldsymbol{x}(t) \in \mathbb{X} \\ \operatorname{dist}(\boldsymbol{x}(t), \mathbb{M}) \underset{t \to T}{\longrightarrow} 0 \end{array} \right\} \tag{15}$$

Remark 5. Definition 1 covers many frameworks, such as:

- Infinite time RoA ($T = \infty$, $\mathbb{X} = \mathbb{R}^n$, often $\mathbb{M} = \{0\}$)
- Maximal positively invariant set ($T = \infty$, $\mathbb{M} = \mathbb{X} \subsetneq \mathbb{R}^n$)
- Constrained finite time RoA ($T < \infty$, $\mathbb{X}$ compact)

We now introduce an infinite dimensional Linear Programming (LP) problem that is related to the constrained finite horizon RoA (see [10] for details):

$$\begin{aligned} W^\star := \inf \ & \int_{\mathbb{X}} w(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \\ \text{s.t. } & v \in C^\infty(\mathbb{R}^{n+1}), w \in C^\infty(\mathbb{R}^n) \\ & w \geq 0 && \text{on } \mathbb{X} && \text{(16a)} \\ & w \geq v(0, \cdot) + 1 && \text{on } \mathbb{X} && \text{(16b)} \\ & \mathcal{L}_{\boldsymbol{f}} v := \partial_t v + \boldsymbol{f}^\top \boldsymbol{\partial}_{\boldsymbol{x}} v \leq 0 && \text{on } \mathbb{\Gamma} && \text{(16c)} \\ & v(T, \cdot) \geq 0 && \text{on } \mathbb{M} && \text{(16d)} \end{aligned}$$

where $\mathbb{\Gamma} := [0,T] \times \mathbb{X} \Subset \mathbb{R}^{n+1}$ denotes a time-state cylinder.

Proposition 1 ([10])**.** *Let (v, w) be feasible for* (16)*. Then,*

$$\begin{aligned} \mathcal{R}_T^{\mathbb{M}} & \subset \{\boldsymbol{x} \in \mathbb{R}^n \ : \ v(0, \boldsymbol{x}) \geq 0\} && \text{(17a)} \\ & \subset \{\boldsymbol{x} \in \mathbb{R}^n \ : \ w(\boldsymbol{x}) \geq 1\} =: \mathbb{L}(w \geq 1) && \text{(17b)} \end{aligned}$$

With Proposition 1, constraint (16a) enforces $w \geq \mathbb{1}_{\mathcal{R}_T^{\mathbb{M}}}$, where $\mathbb{1}_{\mathbb{A}}$ denotes the boolean indicator function of $\mathbb{A} \subset \mathbb{R}^n$ (with value 1 in $\mathbb{A}$ and 0 elsewhere). Moreover, it is proven in [10] that for any minimizing sequence $(v_d, w_d)_{d \in \mathbb{N}}$ for (16), one has $w_d \underset{d \to \infty}{\longrightarrow} \mathbb{1}_{\mathcal{R}_T^{\mathbb{M}}}$ in the sense of $L^1(\mathbb{X})$, so that the volume of the approximation error $\mathbb{L}(w_d \geq 1) \setminus \mathcal{R}_T^{\mathbb{M}}$ converges to 0.

The Moment-SoS hierarchy allows its user to compute such a minimizing sequence, under the following assumptions.

Assumption 1. All considered inputs are polynomial:

1.1. $\boldsymbol{f} \in \mathbb{R}[\mathbf{x}]^n$, so that $v \in \mathbb{R}[\mathrm{t}, \mathbf{x}] \implies \mathcal{L}_{\boldsymbol{f}} v \in \mathbb{R}[\mathrm{t}, \mathbf{x}]$

1.2. $\mathbb{X} = \cap_{i=1}^{m} \mathbb{L}(g_i \geq 0) =: \mathbb{L}(\boldsymbol{g} \in \mathbb{R}_+^m)$ with $\boldsymbol{g} \in \mathbb{R}[\mathbf{x}]^m$

1.3. $\mathbb{M} = \cap_{j=1}^{\ell} \mathbb{L}(h_j \geq 0) = \mathbb{L}(\boldsymbol{h} \in \mathbb{R}_+^{\ell})$ with $\boldsymbol{h} \in \mathbb{R}[\mathbf{x}]^{\ell}$

where $\mathbf{x}$ (resp. t) denotes the dimension n (resp. 1) indeterminate (i.e. identity function, which can be evaluated in any state $\boldsymbol{x} \in \mathbb{R}^n$, resp. time $t \in \mathbb{R}$).

Then, it is possible to work with polynomial Sums-of-Squares (SoS), with the following definitions.

Definition 3. Let $p \in \mathbb{R}[\mathbf{x}]$. Then, considering $g_0 := 1$,

- p is SoS *iff* $p = q_1^2 + \ldots + q_N^2$, $q_1, \ldots, q_N \in \mathbb{R}[\mathbf{x}]$
- $p \in \mathcal{Q}(\boldsymbol{g})$ *iff* $p = s_0\, g_0 + \ldots + s_m\, g_m$, $s_0, \ldots, s_m$ SoS
- $p \in \mathcal{Q}_d(\boldsymbol{g})$ *iff* $p \in \mathcal{Q}(\boldsymbol{g})$ with $\max(\deg s_i\, g_i) \leq d$

Since SoS polynomials are nonnegative by design, it is clear from Definition 3 that any $p \in \mathcal{Q}_d(\boldsymbol{g})$ (resp. $\mathcal{Q}_d(\boldsymbol{h})$) is nonnegative on $\mathbb{X}$ (resp. $\mathbb{M}$), which gives access to a strenghtening of problem (16):

$$
\begin{aligned}
W_d^\star := \inf\ & \int_{\mathbb{X}} w(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \\
\text{s.t. } & v \in \mathbb{R}[\mathrm{t}, \mathbf{x}], w \in \mathbb{R}[\mathbf{x}] \\
& w \qquad\qquad\qquad \in \mathcal{Q}_d(\boldsymbol{g}) && \text{(18a)} \\
& w - v(0, \mathbf{x}) - 1 \quad \in \mathcal{Q}_d(\boldsymbol{g}) && \text{(18b)} \\
& -\mathcal{L}_{\boldsymbol{f}} v \qquad\quad\ \in \mathcal{Q}_d(\boldsymbol{g}, (T - \mathrm{t})\,\mathrm{t}) && \text{(18c)} \\
& v(T, \mathbf{x}) \qquad\quad\ \in \mathcal{Q}_d(\boldsymbol{h}) && \text{(18d)}
\end{aligned}
$$

where $\Gamma = \mathbb{L}((T - \mathrm{t})\,\mathrm{t} \geq 0) \times \mathbb{X} = \mathbb{L}((\boldsymbol{g}, (T - \mathrm{t})\,\mathrm{t}) \in \mathbb{R}_+^{m+1})$.
Problem (18) consists in looking for feasible (v, w) for (16) under the form of polynomials, restricting inequality constraint (16x) into SoS constraint (18x), x=a–d. The advantage of this new problem is that the decision variables are now finite dimensional vectors of coefficients, and the SoS constraints can be recast as LMIs [12]. Thus, assuming knowledge of the moments of the Lebesgue measure on $\mathbb{X}$ (i.e. being able to integrate polynomials on $\mathbb{X}$, e.g. if $\mathbb{X}$ is a ball or a box), one is able to solve (18) on a standard computer, provided that n and d are small enough to make the LMIs tractable.

As the new feasible space is strictly included in the former, there is a relaxation gap: $\forall d \in 2\mathbb{N}$, $W^\star < W_d^\star$. However, it is proved in [10] that, if $\mathbb{X}$ and $\mathbb{M}$ are bounded, $W_d^\star \underset{d \to \infty}{\longrightarrow} W^\star$. Thus, solving instances of (18) gives access to converging outer approximations of the RoA.

This framework comes with several extensions (not detailed here) among which:

- Approximation of a free final time RoA [10]

$$
\mathcal{R}_{[0,T]}^{\mathbb{M}} := \left\{ \boldsymbol{x}(0) \in \mathbb{R}^n \ : \ \begin{array}{l} \forall t \in [0, T],\ \ \boldsymbol{x}(t) \in \mathbb{X} \\ \exists\, t \in [0, T],\ \ \boldsymbol{x}(t) \in \mathbb{M} \end{array} \right\}
$$

- Inner RoA approximation [11]
- RoA approximation for non-polynomial (trigonometric) dynamics [4]
- Maximal positively invariant set approximation [8, 9]

.2 A standard polynomial system

To further illustrate how SOStab is able to reproduce existing results in SoS programming for stability analysis, one can consider a reversed-time Van der Pol oscillator, as in [11]:

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \begin{pmatrix} -2x_2 \\ 0.8x_1 + 10(x_1^2 - 0.21)x_2 \end{pmatrix} \tag{19}$$

The dynamics are polynomial and the stable equilibrium $\boldsymbol{x}^\star$ of the system is at the origin, so that it takes very few lines of code to get interesting results:

```
VdP = SOStab([0;0], [1.1;1.1]);
VdP.dynamics= [-2*VdP.x(2); 0.8*VdP.x(1)
+ 10*(VdP.x(1)^2 - 0.21)*VdP.x(2)];
d = 12;         T = 1;         epsilon = 0.5;
[vol, vc, wc] = VdP.SoS_out(d, T, epsilon);
VdP.plot_roa(1, 2, 'outer');
[vol, vc, wc] = VdP.SoS_in(d, T, epsilon);
VdP.plot_roa(1, 2, 'inner',1);
```

This gives the plot represented in Figure 7, which reproduces results presented in [10, Figure 2] and [11, Figure 3].

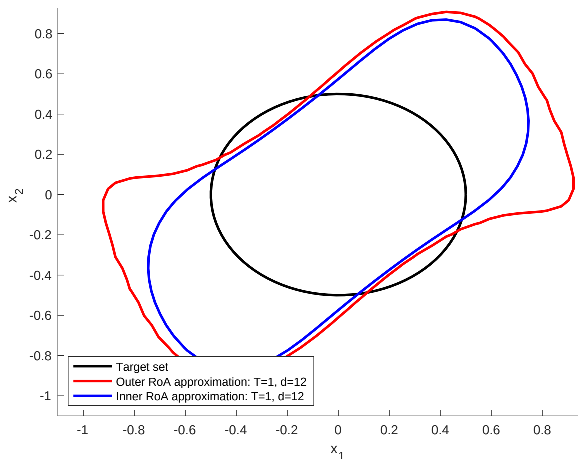

Figure 7: Inner and outer RoA approximations for the Vanderpol system.