Stability Verification Using Sum-of-Squares Programming

Anoop Bhat
Robotics Institute
Carnegie Mellon University
Pittsburgh, United States
agbhat@andrew.cmu.edu

Khai Nguyen
Mechanical Engineering Department
Carnegie Mellon University
Pittsburgh, United States
xkhai@cmu.edu

***Abstract*—We explore sampling-based sum-of-squares (SOS) programming as a technique to verify the stability of nonlinear systems. We provide a review of Lyapunov analysis, traditional SOS stability verification, and sampling-based SOS verification, then apply these methods to two nonlinear systems. We demonstrate that sampling-based SOS verification offers significant advantages over the traditional formulation in terms of problem size and the amount of time required to obtain a solution. The code for this project is available at https://github.com/EpicDuckPotato/final_project_16715.git.**

I. Introduction

Deploying robots into dangerous environments and environments with humans requires ensuring that their operation is safe. One control-theoretic notion of safety is stability–the property that within some range of operating conditions, the robot can be trusted to track desired trajectories or drive itself to a desired operating point. Recently, there has been a great deal of work in verifying stability using sum-or-squares (SOS) programming, which expresses Lyapunov stability conditions as nonnegativity conditions for sets of polynomials [1]. A problem with the traditional SOS approach is its scalabiltiy to higher-dimensional systems, particularly due to its reliance on high-degree Lagrange multiplier polynomials for representing constraints, leading to large semidefinite programs. This is restrictive since there are many intriguing real-world applications that are far larger than 10-15 states, such as mechanical systems made up of several rigid bodies. Various techniques have been used to address this problem, e.g. DSOS and SDSOS programming [2].

In this project, we examine a recent sampling-based method of SOS programming that addresses the scalability problem [3], and compare it to the traditional SOS approach. In II, we review Lyapunov stability analysis, traditional SOS verification, and sampling-based SOS verification. In III, we apply these methods to a Van der Pol oscillator and find that the sampling-based method results in a faster, smaller optimization problem. We also apply the sampling-based method to a N-link cartpole. In IV, we discuss next steps.

II. Methods

A. Lyapunov Stability Analysis

Suppose we have the following autonomous dynamical system

$$\dot{\mathbf{x}} = f(\mathbf{x}) \tag{1}$$

and want to verify the asymptotic stability of the origin $\mathbf{x} = 0$. These dynamics might represent the closed-loop dynamics of a controller attempting to drive the system's state to the origin. Lyapunov analysis proceeds by searching for a Lyapunov function V that satisfies the following two conditions:

$$V(\mathbf{x}) \succ 0 \tag{2}$$

$$\dot{V}(\mathbf{x}) \prec 0 \tag{3}$$

If such a function exists, then the origin $\mathbf{x} = 0$ is asymptotically stable.

For many systems, however, the origin is only stable within some region of attraction (ROA). In Lyapunov analysis, we often write the ROA as the ρ-sublevel set of V, i.e. the origin is stable only when $V < \rho$. Formally,

$$V(\mathbf{x}) \leq \rho \implies \dot{V}(\mathbf{x}) \prec 0 \tag{4}$$

B. Stability Verification via Sum-of-Squares Programming

SOS stability analysis makes the assumption that V and f are polynomial functions of $\mathbf{x}$. In practice, we formulate candidate Lyapunov functions V using the system's total energy or the value function of an LQR controller, which are already polynomial. If the dynamics f are not polynomial, we can approximate them as polynomial using a Taylor expansion. With polynomial V and f, $\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x})$ is a product of polynomials, and thus also polynomial.

For simplicity, suppose we have a positive-definite V, which means we only need to verify (3). With the assumption that $\dot{V}(\mathbf{x})$ is polynomial, a sufficient condition for (3) is that $-\dot{V}(\mathbf{x}) - \epsilon \mathbf{x}^\intercal \mathbf{x}$ is a SOS polynomial, i.e.

$$-\dot{V}(\mathbf{x}) - \epsilon \mathbf{x}^\intercal \mathbf{x} \text{ is SOS} \tag{5}$$

A polynomial $P(\mathbf{x})$ is SOS if it can be written as a sum of squared polynomials, i.e.

$$P(\mathbf{x}) = \sum_i g_i(\mathbf{x})^2 \tag{6}$$

for some polynomials $g_i(\mathbf{x})$. The sufficiency of (5) for (3) can be seen using the following chain of implications, using the fact that all SOS polynomials are non-negative.

$$\begin{aligned} -\dot{V}(\mathbf{x}) - \epsilon\mathbf{x}^\intercal\mathbf{x} \text{ is SOS} &\implies -\dot{V}(\mathbf{x}) - \epsilon\mathbf{x}^\intercal\mathbf{x} \geq 0 & (7)\\ &\implies \dot{V}(\mathbf{x}) \leq -\epsilon\mathbf{x}^\intercal\mathbf{x} & (8)\\ &\implies \dot{V}(\mathbf{x}) \prec 0 & (9)\end{aligned}$$

Note that SOS verification is conservative, since (5) is a sufficient condition for (3), not a necessary one.

We can verify regions of attraction using SOS programming as well, which now requires verifying the implication (4). We incorporate the implication into (5) using the S-procedure:

$$\begin{aligned} -\dot{V}(\mathbf{x}) - \epsilon\mathbf{x}^\intercal\mathbf{x} - \lambda(\mathbf{x})(\rho - V(\mathbf{x})) \text{ is SOS} \quad (10)\\ \lambda(\mathbf{x}) \text{ is SOS} \quad (11)\end{aligned}$$

where λ is a Lagrange multiplier polynomial.

We solve an SOS program by transcribing it into a semi-definite program (SDP). For example, the conditions (10) and (11) correspond to the following SDP:

$$\begin{aligned} &\text{find}_{Q\succeq 0, Q_\lambda \succeq 0}\\ -\dot{V}(\mathbf{x}) - m_\lambda(\mathbf{x})^\intercal Q_\lambda m_\lambda(\mathbf{x})(\rho - V(\mathbf{x})) &= m(\mathbf{x})^\intercal Q m(\mathbf{x})\end{aligned} \tag{12}$$

Here $m(\mathbf{x})$ denotes the standard monomial basis of appropriate degree. The SDP is a convex optimization, and we solve it using Mosek.

Note that the optimization in (10) and (11) is for a fixed value of ρ–that is, it verifies a region of attraction. To find a region of attraction, we perform a binary search over ρ values, where we find the largest possible ρ such that the optimization if feasible (see Algorithm 1).

Algorithm 1: Binary search with SOS program.

```
Input  : Candidate $V$, $\dot{V}$.
Output: ROA $\rho$.
1  initialize $\epsilon$, degree of $\lambda(\mathbf{x})$, initial and lower bound of
     $\rho$, tolerance $\varepsilon$, max_iter;
2  // Find upper bound of $\rho$
3  while is_sos do
4      $\rho \leftarrow 2\rho$;
5      is_sos ← check_sos(Eq. (12));
6  end while
7  upper ← $\rho$;
8  while $\Delta\rho > \varepsilon$ or reach max_iter do
9      is_sos ← check_sos(Eq. (12));
10     if is_sos then
11         lower ← $\rho$; // Increase $\rho$
12     else
13         upper ← $\rho$; // Decrease $\rho$
14     end if
15     $\rho \leftarrow \frac{lower+upper}{2}$;
16     $\rho_{prev} \leftarrow lower$;
17     $\Delta\rho \leftarrow \rho - \rho_{prev}$;
18 end while
19 return $\rho$
```

C. Stability Verification via Sampling-based Sum-of-Squares Programming

Beside the popular SOS programming formulation described above, which is based on inequality implication (4), S-procedure and multipliers, there is also an equality-constrained formulation as follows:

$$\begin{aligned} \max_{\rho, Q\succeq 0, \lambda(\mathbf{x})} \quad & \rho\\ \text{s.t.} \quad & (\mathbf{x}^\intercal\mathbf{x})^d (V(\mathbf{x}) - \rho) - \lambda(\mathbf{x})\dot{V}(\mathbf{x})\\ & \qquad = m(\mathbf{x})^\intercal Q m(\mathbf{x}), \forall \mathbf{x}\end{aligned} \tag{13}$$

The SOS factorization constraint is written explicitly and has a similar form as in (12). However, this formulation does not require the multiplier λ to be SOS, nor does it require the line-search for ROA in Algorithm 1, making the problem simpler.

The $(\mathbf{x}^\intercal\mathbf{x})^d$ term, $d \in \mathbb{R}^+$, ensures that (13) does not always return the trivial solution $\rho = 0$.

An important theorem to find ROA using the formulation above is provided in [3] as follows

Theorem 1: Under the assumption that the Hessian of $\dot{V}$ is negative definite at the origin, the implication condition

$$\dot{V}(\mathbf{x}) = 0 \Rightarrow V(\mathbf{x}) \geq \rho \text{ or } \mathbf{x} = 0 \tag{14}$$

is a sufficient condition for (4).

Define an *algebraic variety* as the set of solutions to a set of polynomial equations. The number of equations in the variety is the number of *components*. The core idea behind sampling-based SOS verification is that instead of solving the optimization problem for all real-valued $\mathbf{x}$, we only solve it at a set of numerical samples $\{\mathbf{x}_i\}$ in the algebraic variety $\mathcal{V} := \{\mathbf{x}|\dot{V}(\mathbf{x}) = 0\}$.

$$\begin{aligned} \max_{\rho, Q\geq 0} \quad & \rho\\ \text{s.t.} \quad & \dot{V}(\mathbf{x}_i) = 0, \forall \mathbf{x}_i\\ & (\mathbf{x}_i^\intercal\mathbf{x}_i)^d (V(\mathbf{x}_i) - \rho) = m^\intercal(\mathbf{x}_i) Q m(\mathbf{x}_i), \forall \mathbf{x}_i\end{aligned} \tag{15}$$

The sampled program (15) is equivalent to the original program (13), if the samples $\{\mathbf{x}_i\}$ are generic. Formally, the genericity condition requires evaluating the basis monomial vector $m(\mathbf{x}_i)$ at each sample $\mathbf{x}_i$, then constructing the matrix M with these monomial vectors as columns, i.e. for a set of p samples, we would have $M = [m(\mathbf{x}_1), \ldots m(\mathbf{x}_p)]$. Then we check the rank of $(M^T M) \circ (M^T M)$, where $\circ$ denotes an Hadamard (elementwise) product. If the matrix is full rank, meaning the samples are unique, but the rank is less than $n(n+1)/2$, where n is the dimension of the state $\mathbf{x}$, then the samples are not generic, and we need to obtain more samples. Details about genericity-checking can be found in [5].

The genericity condition is very important as we need to collect enough good samples or the ROA will be almost zero.

D. Sampling on Algebraic Varieties

Solving (15) requires sampling on an algebraic variety. We implemented two methods for this.

The first method we implemented is valid for varieties with a single component, i.e. a single equation. Suppose $\mathbf{x} \in \mathbb{R}^n$, and we want to sample roots of $\dot{V}(\mathbf{x}) = 0$. Then we sample two vectors $\alpha, \beta \in \mathbb{R}^n$ from a normal distribution, then construct a univariate polynomial in a parameter t as follows:

$$\mathbf{x}(t) = \alpha t + \beta \tag{16}$$

$$\dot{V}(t) = \dot{V}(\mathbf{x}(t)) = \dot{V}(\alpha t + \beta) \tag{17}$$

We then find the roots of the univariate polynomial (17) using numpy's `polyroots` function, which finds roots as the eigenvalues of a companion matrix. The procedure is summarized in 2.

Algorithm 2: Sampling using Polyroots

```
Input  : Scalar-valued function $g : \mathbb{R}^n \to \mathbb{R}$, normal
         distribution parameters $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$,
         upper bound $\mathbf{x}_U$, lower bound $\mathbf{x}_L$, tolerance $\epsilon$
Output: Samples $\{\mathbf{x}_1, \dots \mathbf{x}_p\}$
1 $\alpha, \beta \sim \mathcal{N}(\mu, \Sigma)$
2 $t_1, \dots t_m = \texttt{polyroots}(g(\alpha t + \beta))$
3 $S = \emptyset$
4 for $i \in [m]$ do
5     if $Imag(t_i) \le \epsilon$ and $\mathbf{x}_L \le \alpha t_i + \beta \le \mathbf{x}_U$ then
6         $S \leftarrow S \cup \{\alpha t_i + \beta\}$
7     end if
8 end for
9 return $S$
```

The second method we implemented was Newton's method with an Armijo line search, which applies to varieties with multiple components, e.g. the varieties defined in verification problems with implicit dynamics, as described in II-E. We obtain multiple samples by sampling multiple initial conditions, then running Newton's method to convergence. The pseudocode for this method is shown in 3, where † indicates the Moore-Penrose pseudoinverse.

A simplified description of the sampling-based SOS verification is shown in Algorithm 4. The sample_variety function applies either 4 or 3 repeatedly to obtain new samples. The balancing_V function balances the set by removing close instances which have approximately equal V. The check_genericity function applies the matrix rank test discussed earlier.

E. Verification of Implicit Dynamics

When equation (1) is applied to multibody dynamics, we obtain the following non-polynomial dynamics:

$$\dot{\mathbf{v}} = M(\mathbf{q})^{-1}(\tau(\mathbf{q}, \mathbf{v}) - b(\mathbf{q}, \mathbf{v})) \tag{18}$$

where $\mathbf{q} \in Q$ is the robot's configuration and $\mathbf{v} \in T_{\mathbf{q}}Q$ is the robot's velocity, and $\tau(\mathbf{q}, \mathbf{v})$ is the vector of applied generalized forces. We assume that the applied forces are a

Algorithm 3: Sampling using Newton's Method

```
Input  : Vector-valued function $g : \mathbb{R}^n \to \mathbb{R}^m$, $m \le n$,
         normal distribution parameters $\mu \in \mathbb{R}^n$,
         $\Sigma \in \mathbb{R}^{n \times n}$, tolerance $\epsilon$, line search
         parameters $b$
Output: Sample $\mathbf{x} \in \mathbb{R}^n$
1  $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$
2  while $\frac{1}{2}\|g(\mathbf{x})\|^2 \le \epsilon$ do
3      $J = \frac{\partial g}{\partial \mathbf{x}}$
4      $\Delta \mathbf{x} = -J^\dagger g(\mathbf{x})$
5      $\alpha = 1$
6      while
         $\frac{1}{2}\|g(\mathbf{x} + \alpha \Delta \mathbf{x})\|^2 > \frac{1}{2}\|g(\mathbf{x})\|^2 + b\alpha g(\mathbf{x})^T J \Delta \mathbf{x}$ do
7          $\alpha \leftarrow \frac{1}{2}\alpha$
8      end while
9      x $\leftarrow \mathbf{x} + \alpha \Delta \mathbf{x}$end while
10     return x
11
```

Algorithm 4: Sampling-based SOS program.

```
Input  : Candidate $V$, $\dot{V}$.
Output: ROA $\rho$.
1  initialize $d$;
2  // Find samples
3  $enough\_sample \leftarrow False$;
4  $S = \emptyset$
5  while not enough_sample do
6      // Use numpy polyroots or Newton's
         method
7      $S \leftarrow S \cup$ sample_variety($\dot{V}$);
8      $S \leftarrow$ balancing_V($S, V$);
9      $enough\_sample \leftarrow$ check_genericity($S$);
10 end while
11 calculate $(\mathbf{x}_i^\intercal \mathbf{x}_i)^d, V(\mathbf{x}_i), m(\mathbf{x}_i)$ for $\mathbf{x}_i \in S$
12 // Solve SDP with samples
13 $\rho \leftarrow$ solve(Eq. 15);
14 return $\rho$
```

function of $\mathbf{q}$ and $\mathbf{v}$, e.g. via closed-loop control, to retain autonomous dynamics.

One option to obtain polynomial dynamics is to simply take the Taylor expansion of (18). Another option is to skip the multiplication by the mass matrix, giving the familiar manipulator equation

$$M(\mathbf{q})\dot{\mathbf{v}} + b(\mathbf{q}, \mathbf{v})) = \tau(\mathbf{q}, \mathbf{v}) \tag{19}$$

which represents the dynamics in implicit form.

In the case that we have revolute joints on the robot, $\mathbf{q}$ traditionally contains joint angles θ, so the dynamics terms $M(\mathbf{q})$ and $b(\mathbf{q}, \mathbf{v})$ contain terms $\sin\theta$ and $\cos\theta$, which are not polynomial. However, if we modify our configuration coordinates to contain variables $c \equiv \cos\theta$ and $s \equiv \sin\theta$

instead of θ, with a constraint

$$c^2 + s^2 = 1 \tag{20}$$

we obtain a configuration $\mathbf{q}'$ and dynamics

$$M(\mathbf{q}')\dot{\mathbf{v}} + b(\mathbf{q}', \mathbf{v})) = \tau(\mathbf{q}, \mathbf{v}) \tag{21}$$

The dynamics (21) are almost polynomial, apart from the applied forces $\tau(\mathbf{q}, \mathbf{v})$. For example, consider controlling a one-link manipulator with $\mathbf{q} = \theta \in \mathbb{R}$, driving θ and $\dot{\theta}$ to zero, i.e.

$$\tau(\theta, \dot{\theta}) = -k_p\theta - k_d\dot{\theta} \tag{22}$$

We now need to transform our controller to act on the trigonometric configuration coordinates $\mathbf{q}' = [c, s]$. Using the small angle approximation, $s \approx \theta$, we modify our controller as follows:

$$\tau(c, s, \dot{\theta}) = -k_p s - k_d\dot{\theta} \tag{23}$$

Now the dynamics are polynomial:

$$M(\mathbf{q}')\dot{\mathbf{v}} + b(\mathbf{q}', \mathbf{v})) = \tau(\mathbf{q}', \mathbf{v}) \tag{24}$$

Now we need to verify the stability condition (5) everywhere that the equality constraints (24) and (20) hold, defining $\mathbf{x} = [\mathbf{q}'^\intercal, \mathbf{v}^\intercal]^\intercal$.

In the traditional SOS formulation, this corresponds to adding additional multiplier polynomials, resulting in the following SOS program:

$$-\dot{V}(\mathbf{x}) - \epsilon\mathbf{x}^\intercal\mathbf{x} - \lambda(\mathbf{x})(\rho - V(\mathbf{x})) - \eta(\mathbf{x})^\intercal e(\mathbf{x}) \text{ is SOS} \tag{25}$$

$$\lambda(\mathbf{x}) \text{ is SOS} \tag{26}$$

where $\eta(\mathbf{x})$ is a vector of multiplier polynomials which need not be SOS, and $e(\mathbf{x})$ is the vector of equality constraints (24) and (20) stacked together. With many equality constraints, the program requires many multipliers, resulting in a large SDP.

In the sampling-based version, we simply add more equations to our algebraic variety, i.e.

$$\begin{aligned} \max_{\rho, Q \geq 0} \quad & \rho \\ \text{s.t.} \quad & \dot{V}(\mathbf{x}_i) = 0, \forall \mathbf{x}_i \\ & e(\mathbf{x}_i) = 0, \forall \mathbf{x}_i \\ & (\mathbf{x}_i^\intercal \mathbf{x}_i)^d (V(\mathbf{x}_i) - \rho) = m^\intercal(\mathbf{x}_i) Q m(\mathbf{x}_i), \forall \mathbf{x}_i \end{aligned} \tag{27}$$

We sample on the variety using Newton's method.

III. Results

A. Verification of Van der Pol Oscillator

We verify both standard and sample-based methods derived above by comparing solve time and ROA values compared to *ground truth* which could be found by rolling out closed-loop system simulations.

Van der Pol is a 2 state, degree 3 polynomial system. Its time-reversed model is shown below

$$\begin{aligned} \dot{x}_1 &= -x_2 \\ \dot{x}_2 &= x_1 + x_2(x_1^2 - 1) \end{aligned} \tag{28}$$

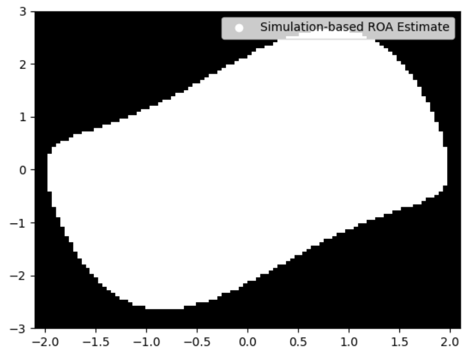

Fig. 1. Van der Pol simulation-based ROA approximation as ground truth.

We first simulate the system on an interesting grid for a certain amount of time. In this way, stable states could be found within some tolerances. The result from simulation-based verification is described in Figure 1. Intuitively, if one needs better quality ROA using simulation, the complexity would increase exponentially with respect to problem dimension, grid size and converging time. This is why SDP programs are preferred for this ROA application. While the standard SOS program is not trivial to set up, the sampling-based one is simpler but in need of an efficient sampling method. The samples must be well-distributed, generic and not close to zeros (trivial solution) to find a non-zero ROA. Both programs are solved with Mosek using a degree 4 candidate V, resulting in good approximations (Figure 2). If the degree of the candidate V is increased, tighter ROA approximations could be found.

$$\begin{aligned} V = 1.8027e^{-6} + 0.28557x_1^2 + 0.0085754x_1^4 + 0.18442x_2^2 \\ +0.016538x_2^4 - 0.34562x_2x_1 + 0.064721x_2x_1^3 \\ +0.10556x_2^2x_1^2 - 0.060367x_2^3x_1 \end{aligned} \tag{29}$$

Details of each program are provided in Table I. The sampling-based SOS program is much smaller (less variables and constraints) than the standard one. With the additional sampling step, it is still 30 times and 100 times faster than the others. This scenario may not show it but theoretically the new approach could yield a more feasible SDP and less conservative ROA. More experiments can be found in [3].

TABLE I
Numerical Comparison of Three Methods for ROA Verification

	Van der Pol		
	Standard SOS	***Sampling SOS***	***Simulation***
num. scalarized var.	141	55	N/A
num. constraints	45	11	N/A
time (sec)	1.04	0.034	4.51
RoA ρ	0.999756	1.009835	1.260139

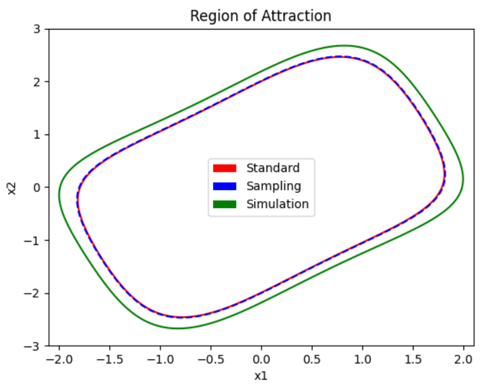

Fig. 2. Van der Pol ROA approximations. Both programs produce qualitatively similar results.

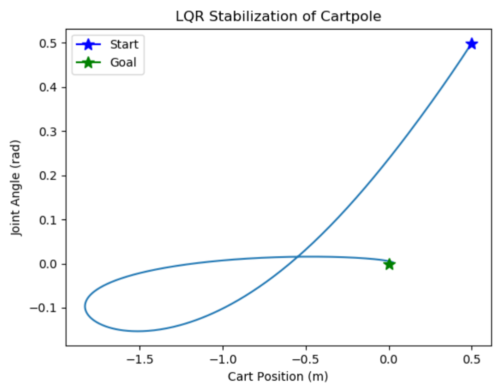

Fig. 3. Cartpole regulation to origin, shown in plane of cart position and joint angle

B. Verification of N-Link Cartpole

We applied sampling-based SOS verification to an N-link cartpole, using the implicit dynamics representation described in II-E. The configuration of the cartpole consists of the position of the cart $p \in \mathbb{R}$, then the joint angles $\theta_i \in \mathbb{R}$, $i \in [N]$.

$$\mathbf{q} = \begin{bmatrix} p \\ \theta_1 \\ \vdots \\ \theta_N \end{bmatrix} \in \mathbb{R}^{1+N} \tag{30}$$

We make the trigonometric substitution $c_i \equiv \cos\theta_i$, $s_i \equiv \sin\theta_i$. Our transformed configuration is

$$\mathbf{q}' = \begin{bmatrix} p \\ c_1 \\ s_1 \\ \vdots \\ c_N \\ s_N \end{bmatrix} \in \mathbb{R}^{1+2N} \tag{31}$$

and we have the following constraint for all $i \in [N]$:

$$c_i^2 + s_i^2 = 1 \tag{32}$$

We assume that the cart, as well as all joints, except for the Nth joint, are actuated, linearize the dynamics in the original coordinates, and design an infinite-horizon LQR controller $\tau(\mathbf{q}, \mathbf{v})$ to drive the full cartpole state, consisting of configuration and velocity, to zero. The trajectory induced by the controller from a perturbed initial condition, in the plane of the cart position and joint angle ($N = 1$ here), is shown in 3. We then make the small-angle approximation $s_i \approx \theta_i$ to obtain a controller in the trigonometric coordinates: $\tau(\mathbf{q}', \mathbf{v})$.

We defined our Lyapunov function as the LQR value function, or cost-to-go, which is quadratic in the state $\mathbf{x}$. We then solved (27), and obtained a ρ of about 1.7, for $N = 1$. We also tried implementing the traditional verification (25), but did not finish debugging.

IV. Conclusion

In this project, we compared a recent sampling-based variant of SOS stability verification to a more traditional form of SOS stability verification, applied to a Van der Pol oscillator. Ground truth for the system was found by using simulation-based method. We found that the sampling-based variant produced a smaller program and took less time to solve. It may also provide less conservative ROA compared with the standard approach. We also applied sampling-based SOS verification to an N-link cartpole. Future directions of research include applying the sampling-based verification to higher-dimensional systems. In particular, systems with contacts would be an interesting application, since they require many multipliers with the traditional SOS formulation [4], creating large SOS programs.

References

[1] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, “LQR-trees: Feedback motion planning via sums-of-squares verification,” The International Journal of Robotics Research, vol. 29, no. 8, pp. 1038–1052, 2010.
[2] A. A. Ahmadi and A. Majumdar, DSOS and SDSOS Optimization: More Tractable Alternatives to Sum of Squares and Semidefinite Optimization. arXiv, 2017. doi: 10.48550/ARXIV.1706.02586.
[3] S. Shen and R. Tedrake, ”Sampling Quotient-Ring Sum-of-Squares Programs for Scalable Verification of Nonlinear Systems,” 2020 59th IEEE Conference on Decision and Control (CDC), 2020, pp. 2535-2542, doi: 10.1109/CDC42340.2020.9304028.
[4] M. Posa, M. Tobenkin and R. Tedrake, ”Stability Analysis and Control of Rigid-Body Systems With Impacts and Friction,” in IEEE Transactions on Automatic Control, vol. 61, no. 6, pp. 1423-1437, June 2016, doi: 10.1109/TAC.2015.2459151.
[5] D. Cifuentes and P. A. Parrilo, “Sampling Algebraic Varieties for Sum of Squares Programs,” SIAM Journal on Optimization, vol. 27, no. 4, pp. 2381–2404, Jan. 2017, doi: 10.1137/15m1052548.