Imperial College London

Department of Aeronautics

PONDS
USER MANUAL

Author: Lino Valencia, Mario A.

Supervisor: Chernyshenko, Sergei I.

August 29, 2019

Acronyms

KSE Kuramoto-Sivashinsky Equation.

MATLAB MATrix LABoratory.

ODE Ordinary Differential Equation.

PDE Partial Differential Equation.

SDP Semi-Definite Program.

SOS Sum of Squares.

SOSP Sum of Squares Program.

YALMIP Yet Another LMI Parser.

Contents

1 What is PONDS?

PONDS (Polynomial Optimisation of Non-linear Dynamical Systems) is an open source MATLAB library for finding upper and lower bounds on long-time averaged polynomial magnitudes in deterministic/stochastic hydrodynamic-type systems of ODEs of the form:

$$\dot{a}_i = N_{ijk}a_ja_k + L_{ij}a_j + B_i + \sqrt{2\epsilon}\sigma_{il}\xi_l, \quad \mathbf{a} \in \mathbb{R}^N, \boldsymbol{\xi} \in \mathbb{R}^m \tag{1}$$

for $i, j = 1, 2, ..., N$ and $l = 1, 2, ..., m$. N_{ijk} is a third order tensor satisfying $N_{ijk}a_ia_ja_k = 0$ (energy conserving non-linearity), L_{ij} is a second order tensor and B_i is a vector. The stochastic vector $\boldsymbol{\xi}(t)$ is the formal derivative of the Wiener process, its components ξ_l are statically independent, delta-correlated, Gaussian-distributed, zero-mean random functions. Matrix $\sigma \in \mathbb{R}^{N\times m}$ relates the effect of each noise component ξ_l on each state variable a_i. The noise intensity can be conveniently tuned by modifying ϵ. For $\epsilon = 0$ we have a deterministic system. PONDS makes possible to find upper U and lower L bounds on infinite-time averaged magnitudes defined as a function of the state vector $\mathbf{a}$ using Sum-of-Squares technique.

A N-dimensional system of the form (1) is normally obtained ater applying a Galerkin expansion to hydrodynamic-type PDEs, such as Navier-Stokes equations and Kuramoto-Sivashinsky equation. It is also common practice to reduce those N-dimensional systems to n-dimensional uncertain systems with $n < N$. PONDS also allows to find bounds for such uncertain systems by first reducing the original N-dimensional system to a n-dimensional uncertain system of the form

$$\begin{gathered}
\frac{d\hat{a}_x}{dt} = \hat{N}_{xyz}\hat{a}_y\hat{a}_z + \hat{L}_{xy}\hat{a}_y + \hat{B}_x + \Theta_x, \quad x, y = 1, 2, ..., n \\
\frac{dq^2}{dt} = -\hat{\mathbf{a}}^T \cdot \boldsymbol{\Theta} + \Gamma \\
\Gamma \leq \kappa q^2, \quad \kappa \in \mathbb{R}^- \\
||\boldsymbol{\Theta}||^2 \leq P(\mathbf{a}, q) = c_1q^2 + c_2q^2||\hat{\mathbf{a}}||^2 + c_3q^4, \quad c_1, c_2, c_3 \in \mathbb{R}^+
\end{gathered}$$

where $(\hat{a}_1, \hat{a}_2, ..., \hat{a}_n, q^2)$ is the new state vector and, $\boldsymbol{\Theta}$ and Γ are uncertain variables. This reduction is done thank to an already existing toolbox named UODESys [1]. PONDS functionality is based on Sum-of-Squares technique. The interested reader may refer to [2] and [3] for an introduction to Sum-of-Squares technique.

2 Requirements and installation

In order to use PONDS, you will need:

- **YALMIP**. YALMIP automates the conversion from SOSPs to SDPs. It calls an SDP solver and converts the SDP solution back to the solution of the original SOSP. YALMIP can be downloaded from here. Note that MATLAB 5.2 or earlier versions are not supported by YALMIP.

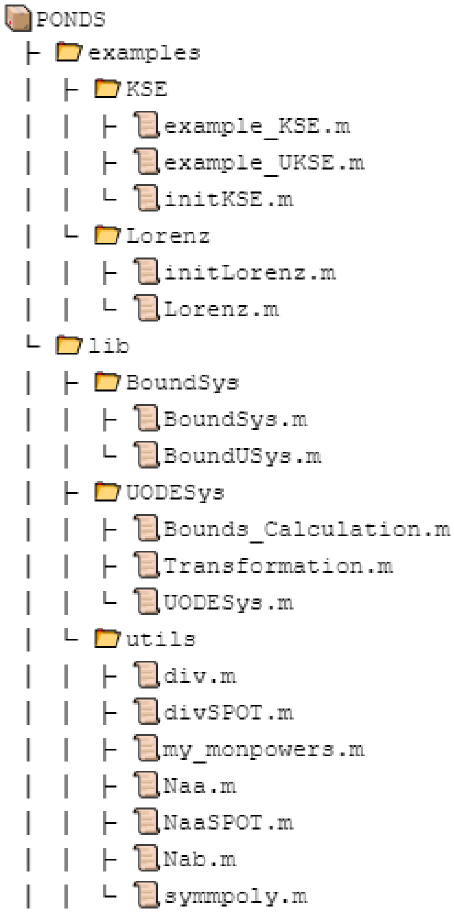

Figure 1: PONDS file tree.

- **SPOTLess**. SPOTLess is **not strictly required**, but can be used as an alternative to YALMIP when finding bounds with functions `BoundSys()` and `BoundUSys()`. It is another SOSP solver similar to YALMIP. SPOTLess is more complex to use than YALMIP, but at the same time is considerably faster for high dimensional problems and high degrees of the auxiliary functionals. It can be downloaded from here. To install SPOTLess simply run the file `spot_install.m`.

- An SDP solver, such as Mosek and SeDuMi. Only this two SDP solvers can be used by `BoundSys()` and `BoundUSys()` if SPOTLess is selected instead of YALMIP when calling these functions.

PONDS can be downloaded or cloned from github:

https://github.com/aeroimperial-optimization/PONDS

Figure 1 shows PONDS file tree. Folder `examples` contains examples of usage and `lib` contains the scripts that constitute PONDS. Remember to add the folder `lib` to MATLAB's search path: `addpath(genpath('./lib'))`. Inside `lib`; `UODESys`, `BoundSys` and `Operations` folders are located. `BoundSys` includes the new functions developed for finding bounds. For its part, `Operations` contains different functions used by `BoundSys()` and `BoundUSys()`.

3 How to use PONDS

A description of the functions available in PONDS is given here.

3.1 UODESys

UODESys (Uncertain Ordinary Differential Equation System) is a free, third-party MATLAB toolbox for derivation of n-dimensional uncertain systems from N-dimensional ODE systems with $N > n$. UODESys was developed by Lakshmi M. [1] to enable future research on uncertain quadratic dynamical systems, such as Navier-Stokes equations and Kuramoto-Sivashinsky equation. It can be downloaded from

https://github.com/aeroimperial-optimization/UODESys

UODESys is already included in PONDS, so it is no required to downloaded it. Basically, UODESys first performs a transformation to energy stability axes and then reduces the dimension of the original quadratic system from N to n, introducing the uncertain variables $\Gamma \in \mathbb{R}$ and $\boldsymbol{\Theta} \in \mathbb{R}^n$. The following diagram illustrates the process:

$$\dot{a}_i = N_{ijk}a_j a_k + L_{ij}a_j + B_i, \quad i, j = 1, 2, ..., N$$

$$\Downarrow$$

$$\frac{d\hat{a}_x}{dt} = \hat{N}_{xyz}\hat{a}_y\hat{a}_z + \hat{L}_{xy}\hat{a}_y + \hat{B}_x + \Theta_x, \quad x, y = 1, 2, ..., n$$
$$\frac{dq^2}{dt} = -\hat{\mathbf{a}}^T \cdot \mathbf{\Theta} + \Gamma$$
$$\Gamma \leq \kappa q^2, \quad \kappa \in \mathbb{R}$$
$$||\mathbf{\Theta}||^2 \leq P(\mathbf{a}, q) = c_1 q^2 + c_2 q^2 ||\hat{\mathbf{a}}||^2 + c_3 q^4, \quad c_1, c_2, c_3 \in \mathbb{R}^+$$

UODESys is already included in PONDS for several reasons. First, it is required by function `BoundUSys`, and second, the user may also want to use it in its own code. If you are interested in using UODESys please refer to UODESys manual, it can be found on

https://github.com/aeroimperial-optimization/UODESys/blob/master/docs/manual.pdf

To facilitate the use of UODESys, a function named `UODESys` has been included in PONDS. This function contains the original code in the scripts `INPUT.m`, `Variable_Transformation.m` and `Bounds_Calculation.m` of UODESys toolbox, and takes the following input arguments:

- `n`: size of the n-dimensional uncertain system to be obtained.
- `g`: vector $[\gamma_1, \gamma_2, \gamma_3]$.
- `input_file`: `.mat` file containing `N_ijk`, `L_ij`, `B_i`.
- `output_file`: `.mat` file containing UODESys outputs.
- `solver`: SDP solver to be used by UODESys.
- `verbose`: `0` for no displays of the SDP solver, `1` to activate displays.

The outputs of `UODESys` are the same as specified by UODESys toolbox manual.

3.2 BoundSys() - Bounds for N-dimensional deterministic/stochastic systems

Function for computing upper and lower bounds on the desired long-time averaged polynomial magnitude in a deterministic/stochastic N-dimensional system of the form

$$\dot{a}_i = N_{ijk}a_j a_k + L_{ij}a_j + B_i + \sqrt{2\epsilon}\sigma_{il}\xi_l, \quad \mathbf{a} \in \mathbb{R}^N, \boldsymbol{\xi} \in \mathbb{R}^m \tag{2}$$

for $i, j = 1, 2, ..., N$ and $l = 1, 2, ..., m$. The stochastic vector $\boldsymbol{\xi}(t)$ is the formal derivative of the Wiener process, its components ξ_l are statically independent, delta-correlated, Gaussian-distributed, zero-mean random functions. Matrix $\sigma \in \mathbb{R}^{N\times m}$ relates the effect of each noise component ξ_l on each state variable a_i. The noise intensity can be conveniently tuned by modifying ϵ. For $\epsilon = 0$ we have the original deterministic system.

3.2.1 Mathematical formulation

In a more practical way, equation (2) can be written as

$$da_i = (N_{ijk}a_j a_k + L_{ij}a_j + B_i)dt + \sigma_{il}N_i(0, 2\epsilon dt) \tag{3}$$

where $N_i(0, 2\epsilon dt)$ denotes a vector whose N components are independent normal functions with zero mean and a variance of $2\epsilon dt$. This equation can be numerically solved using a fixed-step temporal scheme, such as Runge-Kutta order 5, however this may imply extremely large computational time, specially for high dimensional problems. If one is only interested on the steady value of a particular magnitude, it is much more convenient to look for tight bounds for such value using SOS techniques.

In [2] and [3] SOSPs where obtained for finding bounds on long time averaged magnitudes in system (2). It is assumed that the system has reached statical equilibrium and satisfies Fokker-Planck equation

$$\nabla \cdot (\epsilon D\nabla\rho - \mathbf{f}\rho) = 0, \quad \int_{\mathbb{R}^N} \rho(\mathbf{a})d\mathbf{a} = 1, \quad D := \sigma\sigma^T \tag{4}$$

where $\rho(\mathbf{a}) \geq 0$ is the probability density of the trajectories. It is also assumed that $\rho(\mathbf{a})$ decays exponentially at infinity. The SOSPs proposed in [3] to find upper U and lower L bounds are respectively

$$\min_V U \quad s.t. \;\; \mathcal{D}_u := U - \epsilon\nabla\cdot(S\nabla V) - \mathbf{f}\cdot\nabla V - \phi \in \Sigma \tag{5}$$

$$\max_V L \quad s.t. \;\; \mathcal{D}_l := \epsilon\nabla\cdot(S\nabla V) + \mathbf{f}\cdot\nabla V + \phi - L \in \Sigma \tag{6}$$

where $S = \sigma\sigma^T$. The auxiliary functional $V(\mathbf{a})$ has been introduced in equations (5) and (6) . It is defined as

$$V(\mathbf{a}) := c(\mathbf{a}^T\mathbf{a})^d + p_{2d-1}(\mathbf{a}) \tag{7}$$

where $p_{2d-1}(\mathbf{a})$ is a $(2d-1)$-degree polynomial function of $\mathbf{a}$ and d is half the degree of the auxiliary functional. The coefficients of $p_{2d-1}(\mathbf{a})$ and constant c are, together with U or L, the decision parameters to be determined when solving the SOSPs (6) and (5).

It is convenient to take into account the possible symmetries of our problem, whenever they exist, since this can reduce an order of magnitude the computing time. Basically, in case

$\mathbf{f}(\Lambda\mathbf{a}) = \Lambda\mathbf{f}(\mathbf{a})$ and $\phi(\Lambda\mathbf{a}) = \phi(\mathbf{a})$, where

$$\Lambda = \begin{pmatrix} \pm1 & & & & \\ & \pm1 & & & \\ & & \pm1 & & \\ & & & \ddots & \\ & & & & \pm1 \end{pmatrix}$$

there exist an optimal V satisfying $V(\Lambda\mathbf{a}) = V(\mathbf{a})$ [4, p.23-25]. Thus, if V is selected such that $V(\Lambda\mathbf{a}) = V(\mathbf{a})$ for every value of the decision variables, then we can conclude that $\mathcal{D}$ also satisfies $\mathcal{D}(\Lambda\mathbf{a}) = \mathcal{D}(\mathbf{a})$. Under this circumstances, the SOS decomposition for $\mathcal{D}$ can be written as

$$\mathcal{D}_u(\mathbf{a}) = \mathbf{h}(\mathbf{a})^T Q\mathbf{h}(\mathbf{a}) = [\mathbf{h}_1(\mathbf{a})^T, \mathbf{h}_2(\mathbf{a})^T] \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \begin{bmatrix} \mathbf{h}_1(\mathbf{a}) \\ \mathbf{h}_2(\mathbf{a}) \end{bmatrix} = \mathbf{h}_1(\mathbf{a})^T Q_1 \mathbf{h}_1(\mathbf{a}) + \mathbf{h}_2(\mathbf{a})^T Q_2 \mathbf{h}_2(\mathbf{a})$$

where $\mathbf{h}_1(\mathbf{a})$ contains the symmetric monomials ($\mathbf{h}_1(\Lambda\mathbf{a}) = \mathbf{h}_1(\mathbf{a})$) and $\mathbf{h}_2(\mathbf{a})$ contains the anti-symmetric monomials ($\mathbf{h}_2(\Lambda\mathbf{a}) = \Lambda\mathbf{h}_2(\mathbf{a})$). Solving two SDPs, one for Q_1 and another one for Q_2, is computationally more efficient than directly looking for Q [5].

3.2.2 Syntax

The input arguments for `BoundSys` are, in this order:

- `bound`: `'U'` to compute upper bound, `'L'` to compute lower bound.
- `f`: `.mat` file containing `N_ijk`, `L_ij` and `B_i`. `N_ijk` must satisfy $N_{ijk}a_ia_ja_k = 0$
- `magnitude`: user defined function for the magnitude $\phi(\mathbf{a})$ to be bounded.
- `d`: Half the degree of the auxiliary function.
- `epsilon` (optional argument, by default `epsilon = 0`): ϵ in equation (2), i.e., noise intensity. If $\epsilon = 0$ the deterministic problem is solved, avoiding to compute $\epsilon\nabla\cdot(S\nabla V)$.
- `sigma` (optional argument, by default `sigma = eye(N)`): σ in equation (2).
- `verbose` (optional argument, by default `verbose = 0`): `0` for no displays of the SDP solver, `1` to activate displays.
- `symmetries` (optional argument, by default `symmetries = 0`): diagonal of matrix Λ, provided as a row vector. The value of each component must be 1 0r -1. If `symmetries = 0`, no symmetry reduction is applied.
- `SOSPsolver` (optional argument, by default `SOSPsolver = 'yalmip'`): SOSP solver. YALMIP (`'yalmip'`) and SPOTLess (`'spotless'`) are available.

- `SDPsolver` (optional argument, by default `SDPsolver = 'mosek'`): SDP solver. Check here available SDP solver for YALMIP. Only SeDuMi (`'sedumi'`) and Mosek (`'mosek'`) have been implemented for `BoundSys` with SPOTLess as SOSP solver.

A descriptive list with `BoundSys` inputs can be obtained by typing `help BoundUSys` in MATLAB. The outputs for `BoundSys` are:

- `U`: bound for long time averaged `magnitude` (U or L).
- `res`: residual after solving the SDP computed as $||\texttt{coeff}(\mathcal{D}(\mathbf{a}) - \mathbf{h}^T(\mathbf{a})Q\mathbf{h}(\mathbf{a}))||_\infty$.
- `sol`: structure containing YALMIP or SPOTLess outputs.

Examples: Lorenz.m, example_KSE.m.

3.3 BoundUSys() - Bounds for n-dimensional deterministic/stochastic uncertain systems

Function for computing upper and lower bounds on the desired long time averaged magnitude in a deterministic/stochastic N-dimensional system of the form (2). Bounds are not directly computed for (2), but for a n-dimensional uncertain system obtained from (2). Alternatively, if matrix `sigma` (σ in equation (2)) is n-dimensional, then noise is added to the n-dimensional uncertain system obtained from the N-dimensional deterministic system (equation (2) with $\epsilon = 0$).

3.3.1 Mathematical formulation

The procedure here used for finding bounds on long time average magnitudes in stochastic uncertain systems is described in [6]. The noise is originally added to a N-dimensional system, obtaining equation (2). Then, this system is reduced to a corresponding n-dimensional uncertain system. Thus, we need to find first how does the noise term enter in the uncertain system. We will assume that the N-dimensional dynamical system has already been transformed to the system of coordinates defined by the eigenvectors of the energy stability problem. For its part, the noise term in the new system of coordinates is obtained by simply premultiplying $\sqrt{2\epsilon}\sigma\boldsymbol{\xi}$ with T^{-1}. The columns of the transformation matrix $T \in \mathbb{R}^{N\times N}$ consist of the eigenvectors of the energy stability problem (i.e., the eigenvectors of matrix $2(L_{ij} + L_{ji})$) sorted in decreasing order of the associated eigenvalues.

Let's begin by rewriting equation (2) into its indicial form

$$\dot{a}_i = N_{ijk}a_ja_k + L_{ij}a_j + \sqrt{2\epsilon}\sigma_{il}\xi_l, \qquad i,j,k = 1,2,...,N;\ l = 1,2,...,m \tag{8}$$

Notice that there is an implicit summation over j, k and l. Now we introduce vectors

$$\mathbf{b} := (a_1, a_2, ..., a_n)^T, \quad \mathbf{c} := (a_{n+1}, a_{n+2}, ..., a_N)^T$$

$\mathbf{b}$ is the state vector of the n-dimensional uncertain system (containing the most unstable states) and $\mathbf{c}$ is a vector containing the $(N-n)$-residual states (the most stable ones). Introducing these definitions in (8) we can obtain an equation for $\dot{\mathbf{b}}$,

$$\dot{b}_i = \sum_{j,k=1}^{n} N_{ijk}b_jb_k + \sum_{j=1}^{n} L_{ij}b_j + \sum_{j,k=1}^{N-n} N_{i,n+j,n+k}c_jc_k + \sum_{\substack{i,j=1}}^{\substack{j=n\\k=N-n}} N_{i,j,n+k}b_jc_k + \sum_{j,k=1}^{\substack{j=N-n\\k=N}} N_{i,n+j,k}c_jb_k + \sum_{j=1}^{N-n} L_{i,n+j}c_j + \sqrt{2\epsilon}\sigma_{il}\xi_l \tag{9}$$

for $i = 1, 2, ..., n$ and $l = 1, 2, ..., m$. Summations over indices j and k are now made explicit for clarity. After identifying the term $\boldsymbol{\Theta}(\mathbf{b}, \mathbf{c})$, given by

$$\Theta_i = \sum_{j,k=1}^{N-n} N_{i,n+j,n+k}c_jc_k + \sum_{i,j=1}^{\substack{j=n\\k=N-n}} N_{i,j,n+k}b_jc_k + \sum_{j,k=1}^{\substack{j=N-n\\k=N}} N_{i,n+j,k}c_jb_k + \sum_{j=1}^{N-n} L_{i,n+j}c_j \tag{10}$$

equation (9) can be rewritten as

$$\dot{b}_i = f_i(\mathbf{b}) + \Theta_i(\mathbf{b}, \mathbf{c}) + \sqrt{2\epsilon}\sigma_{il}\xi_l, \quad i, j, k = 1, 2, ..., n; \quad l = 1, 2, ..., m \tag{11}$$

An equation for $\dot{q}^2$ is also needed in our problem formulation. Scalar q^2 is the energy associated to the residual terms and is given by $q^2 := ||\mathbf{c}||^2/2$. Multiplying equation (8) by a_i and summing over i yields

$$\sum_{i=1}^{N} a_i\dot{a}_i = \sum_{i,j,k=1}^{N} N_{ijk}a_ia_ja_k + \sum_{i,j=1}^{N} L_{ij}a_ia_j + \sum_{i=1}^{N} B_ia_i + \sum_{i=1}^{N} \sqrt{2\epsilon}\sigma_{il}a_i\xi_l \tag{12}$$

Similarly, if we reintroduce $\mathbf{a}$ in (11), multiply it by a_i and sum over i, we obtain

$$\sum_{i=1}^{n} a_i\dot{a}_i = \sum_{i,j,k=1}^{n} N_{ijk}a_ia_ja_k + \sum_{i,j=1}^{n} L_{ij}a_ia_j + \sum_{i=1}^{n} B_ia_i + \sum_{i=1}^{n} a_i\Theta_i + \sum_{i=1}^{n} \sqrt{2\epsilon}\sigma_{il}a_i\xi_l \tag{13}$$

Subtracting (13) from (12) and taking into account that $\sum_{i,j,k=1}^{N} N_{ijk}a_ia_ja_k = 0$ and $\sum_{i,j,k=1}^{n} N_{ijk}a_ia_ja_k = 0$, we get

$$\dot{q}^2 = -\sum_{i,j=1}^{n} a_i\Theta_i + \sum_{i,j=n+1}^{N} L_{ij}a_ia_j + \sum_{\substack{i=1\\j=n+1}}^{\substack{i=n\\j=N}} L_{ij}a_ia_j + \sum_{\substack{j=1\\i=n+1}}^{\substack{j=n\\i=N}} L_{ij}a_ia_j + \sum_{i=n+1}^{N} B_ia_i + \sum_{i=n+1}^{N} \sqrt{2\epsilon}\sigma_{il}a_i\xi_l \tag{14}$$

This equation can be rewritten as

$$\dot{q}^2 = -\sum_{i=1}^{n} b_i\Theta_i(\mathbf{b}, \mathbf{c}) + \Gamma(\mathbf{c}) + \chi(\mathbf{b}, \mathbf{c}) + \sum_{i=n+1}^{N} B_ia_i + \delta(\mathbf{c}), \quad l = 1, 2, ..., m \tag{15}$$

where $\Gamma(\mathbf{c})$, $\chi(\mathbf{b}, \mathbf{c})$ and $\delta(\mathbf{c})$ are defined as

$$\Gamma(\mathbf{c}) = \sum_{i,j=1}^{N-n} L_{n+i,n+j}c_ic_j \tag{16}$$

$$\chi(\mathbf{b},\mathbf{c}) = \sum_{\substack{i=n \\ j=N-n \\ i,j=1}} L_{i,n+j} b_i c_j + \sum_{\substack{i=N-n \\ j=n \\ i,j=1}} L_{n+i,j} c_i b_j \tag{17}$$

$$\delta(\mathbf{c}) = \sum_{i=1}^{N-n} \sqrt{2\epsilon}\sigma_{n+i,l} c_i \xi_l, \qquad l = 1, 2, ..., m \tag{18}$$

Since we are working in the energy stability axes $\chi = 0$ [7], and the equation for $\dot{q}^2$ reads

$$\dot{q}^2 = -b_i \Theta_i(\mathbf{b},\mathbf{c}) + \Gamma(\mathbf{c}) + \delta(\mathbf{c}) + \sum_{i=1}^{N-n} B_{n+i} c_i, \quad i = 1, 2, ..., n;\ l = 1, 2, ..., m \tag{19}$$

Given that the new state vector comprises $\mathbf{b}$ and q^2, but not $\mathbf{c}$; variables depending on $\mathbf{c}$ are said to be uncertain variables. Noise enters into the equation for $\dot{q}^2$ as an uncertain variable δ. Uncertain variables in equations (11) and (19) can be easily bounded. Bounds for $\mathbf{\Theta}$ and Γ are given by [7]

$$\Gamma \leq \kappa q^2, \quad \kappa \in \mathbb{R}^- \tag{20}$$

$$||\mathbf{\Theta}||^2 \leq P(\mathbf{a}, q) = c_1 q^2 + c_2 q^2 ||\mathbf{a}||^2 + c_3 q^4, \quad c_1, c_2, c_3 \in \mathbb{R}^+ \tag{21}$$

For its part, δ can be upper bounded applying Schwarz's theorem to equation (18),

$$|\delta| \leq ||\sqrt{2\epsilon}\sigma^* \boldsymbol{\xi}|| \cdot ||\mathbf{c}|| = 2\sqrt{\epsilon}||\sigma^* \boldsymbol{\xi}|| \cdot |q|$$

where σ^* is the submatrix of σ containing the last $N - n$ rows. Introducing constant $\mu := 2\sqrt{\epsilon}||\sigma^* \boldsymbol{\xi}||$, the bound for δ reads

$$\delta \leq \mu |q| \tag{22}$$

Equations (11), (19), (21), (20) and (22) define the n-dimensional stochastic uncertain system that we were looking for. Now we can proceed to find bounds on long time averaged magnitudes. Fokker-Plank equation applies to equations (11) and (19) when the uncertain system has reached statical equilibrium,

$$\nabla \cdot \left(\epsilon D \nabla \rho - \begin{bmatrix} \mathbf{f} + \mathbf{\Theta} \\ \dot{q}^2 \end{bmatrix} \rho \right) = 0, \quad \int_{\mathbb{R}^N} \rho(\mathbf{a}) d\mathbf{a} = 1, \quad S := \left(\begin{array}{c:c} \sigma_{1:n,1:m}\, \sigma^T_{1:n,1:m} & 0_{1:n,1} \\ \hdashline 0_{1,1:m} & 0 \end{array} \right) \tag{23}$$

With this expression for S we are taking into account that the noise term does not appear explicitly in equation (19). Taking into account equation (23) and assuming $\partial V / \partial q^2 \geq 0$, one can arrive to the following condition [6]

$$\mathcal{D}(\mathbf{a}, q^2) = U - \phi - \frac{\partial V}{\partial b}(\mathbf{f} + \Theta) - \frac{\partial V}{\partial (q^2)}(\dot{q^2}) - \epsilon \nabla \cdot (S \nabla V) \geq 0, \quad \forall \mathbf{b} \in \mathbb{R}^n, \forall q^2 \in \mathbb{R} \tag{24}$$

what can be proved to lead to $W(z_0, \mathbf{z}, \mathbf{a}, q^2) \geq 0, \forall z_0, \mathbf{z}, \forall \mathbf{a}, \forall q^2$ [6] with

$$W = -(P(\mathbf{a}, q^2) z_0^2 + \mathbf{z}^T \mathbf{z}) \left(\frac{\partial V}{\partial \mathbf{a}} \mathbf{f} + \frac{\partial V}{\partial (q^2)} \kappa q^2 + \frac{\partial V}{\partial (q^2)} \mu |q| + \phi + \epsilon \nabla \cdot (S \nabla V) - U \right) - 2P(\mathbf{a}, q^2) z_0 \left(\frac{\partial V}{\partial \mathbf{a}} - \frac{\partial V}{\partial q^2} \mathbf{a}^T \right) \mathbf{z} \tag{25}$$

z and z_0 are new independent variables.

From this condition we can derive the corresponding SOSP to compute an upper bound U,

$$\min_V U \quad s.t. \quad \frac{\partial V}{\partial q^2} \in \Sigma, \;\; W(z_0, \mathbf{z}, \mathbf{a}, q^2) \in \Sigma \tag{26}$$

Similarly, a lower bound can be found solving the SOSP

$$\max_V L \quad s.t. \quad \frac{\partial V}{\partial q^2} \in \Sigma, \;\; -W(z_0, \mathbf{z}, \mathbf{a}, q^2) \in \Sigma \tag{27}$$

However, there is a practical drawback in this formulation. δ in equation (19) can take positive values given that $\mu > 0$. Hence, if ϵ is not small enough, it can happen that $\dot{q}^2 \geq 0$. Being the uncertain system defined by (11), (19), (21), (20) and (22) unstable. An alternative is to first reduce the N-dimensional deterministic system

$$\dot{a}_i = N_{ijk} a_j a_k + L_{ij} a_j + B_i, \quad i, j = 1, 2, ..., N$$

to the following n-dimensional uncertain system, also deterministic

$$\frac{d\hat{a}_x}{dt} = \hat{N}_{xyz} \hat{a}_y \hat{a}_z + \hat{L}_{xy} \hat{a}_y + \hat{B}_x + \Theta_x, \quad x, y = 1, 2, ..., n \tag{28}$$

$$\frac{dq^2}{dt} = -\hat{\mathbf{a}}^T \cdot \mathbf{\Theta} + \Gamma \tag{29}$$

$$\Gamma \leq \kappa q^2, \quad \kappa \in \mathbb{R} \tag{30}$$

$$||\mathbf{\Theta}||^2 \leq P(\mathbf{a}, q) = c_1 q^2 + c_2 q^2 ||\hat{\mathbf{a}}||^2 + c_3 q^4, \quad c_1, c_2, c_3 \in \mathbb{R}^+ \tag{31}$$

Thus, noise is only being added to the most unstable modes.

In, this case equation (25) is rewritten as

$$W = -(P(\mathbf{a}, q^2) z_0^2 + \mathbf{z}^T \mathbf{z}) \left(\frac{\partial V}{\partial \mathbf{a}} \mathbf{f} + \frac{\partial V}{\partial (q^2)} \kappa q^2 + \phi + \epsilon \nabla \cdot (S \nabla V) - U \right) - 2P(\mathbf{a}, q^2) z_0 \left(\frac{\partial V}{\partial \mathbf{a}} - \frac{\partial V}{\partial q^2} \mathbf{a}^T \right) \mathbf{z} \tag{32}$$

The SOSP to determine U and L are formally the same that (26) and (27) respectively.

3.3.2 Syntax

The input arguments for `BoundUSys` are, in this order:

- `bound`: `'U'` to compute upper bound, `'L'` to compute lower bound.
- `f`: `.mat` file containing `N_ijk`, `L_ij` and `B_i`. `N_ijk` must satisfy $N_{ijk} a_i a_j a_k = 0$
- `UODESYS_input`: structure containing input arguments for UODESys `{n,[g1,g2,g3],output_file}`. `n` is the dimension of the uncertain system, `[g1,g2,g3]` are used in the optimisation problem solved by UODESys to find c_1, c_2 and c_3 (it seems that `[1,1,1]` produces satisfactory results). `output_file` is the name of the `.mat` file were the outputs from UODESys will be stored. **Important:** if file `output_file` already exists, `BoundUSys` will not call `UODESys` and the data in the exiting file will be used to save computing save.

- `magnitude`: user defined function for the magnitude $\phi(\mathbf{a}, q^2)$ to be bounded.
- `d`: Half the degree of the auxiliary function.
- `epsilon` (optional argument, by default `epsilon = 0`): ϵ in equation (2), i.e., noise intensity. If $\epsilon = 0$ the deterministic problem is solved instead.
- `sigma` (optional argument, by default `sigma = eye(n)`): it corresponds to σ. If `sigma` has size $N \times N$, then W is considered as given by equation (25). If `sigma` has size $n \times n$, noise is added to the already reduced uncertain system and W is is given by (32).
- `verbose` (optional argument, by default `verbose = 0`): `0` for no displays of the SDP solver, `1` to activate displays.
- `SOSPsolver` (optional argument, by default `SOSPsolver = 'yalmip'`): SOSP solver. YALMIP (`'yalmip'`) and SPOTLess (`'spotless'`) are available.
- `SDPsolver` (optional argument, by default `SDPsolver = 'mosek'`): SDP solver. Check here available SDP solver for YALMIP. Only SeDuMi (`'sedumi'`) and Mosek (`'mosek'`) have been implemented for `BoundSys` with SPOTLess as SOSP solver.

A descriptive list with `BoundUSys` inputs can be obtained by typing `help BoundUSys` in MATLAB. The outputs for `BounUdSys` are:

- `U`: bound for long time averaged `magnitude` (U or L).
- `res`: residual after solving the SDP computed as the infinity norm of ($||\texttt{coeff}(\mathcal{D}(\mathbf{a}, q^2) - \mathbf{h}^T(\mathbf{a}, q^2)Q\mathbf{h}(\mathbf{a}, q^2)||_\infty, ||\texttt{coeff}((\partial V/\partial q^2) - \mathbf{v}_2^T(\mathbf{a}, q^2)R\mathbf{v}_2(\mathbf{a}, q^2))||_\infty$).
- `sol`: structure containing YALMIP or SPOTLess outputs.

Examples: example_UKSE.m.

3.4 Examples

3.4.1 Lorenz.m - Lorenz attractor

Files `Lorenz.m` and `initLorenz.m` located in `./examples/Lorenz` are required to run this example.

Lorenz attractor is given given by

$$\begin{array}{rcl} \dot{x} & = & \sigma(y - x) \\ \dot{y} & = & x(\rho - z) - y \\ \dot{x} & = & xy - \beta z \end{array} \tag{33}$$

In this example We look for an upper bound U on long time averaged energy dissipation, $\phi(x,y,z) := \sigma x^2 + y^2 + \beta z^2$, for various values of ρ. The code in `Lorenz.m` is included below.

In line 8 a folder named `data` is created. Here, the `.mat` files containing tensors N_{ijk}, L_{ij} and B_i, obtained by running `initLorenz.m` are stored. Although is no needed to create such folder, it is convenient when the amount of `.mat` files is large. From lines 18 to 24, we specify that: we look for an upper bound, $deg(V) = 8$, no noise is to be added to Lorenz attractor, we want displays from the SDP solver, no symmetry reductions is to be applied and to use YALMIP together with Mosek as SDP solver. A rescaling of 20 is being applied to the state vector **x**. Thus, variable `magnitude` corresponds to the dissipation multiplied by the square of the scaling factor. In line 35, function `initLorenz` is used to generate tensors N_{ijk}, L_{ij} and B_i and stores them in the `.mat` file designated by variable `f`. Finally, line 37 call `BoundSys` with all the input arguments given explicitly.

Listing 1: `Lorenz.m`

```
1  % Compute dissipation in Lorenz attractor with no noise
2  
3  % Written by Mario Lino Valencia (September 2019)
4  % Imperial College London - Department of Aeronautics
5  
6  clear, clc;
7  
8  mkdir data; % Create folder to store N_ijk, L_ij and B_i
9  
10 % Value of the parameters of Lorenz attractor
11 rho = 0:5:50;
12 beta  = 8/3;
13 sigma = 10;
14 
15 rescaling = 20; % Rescaling factor
16 
17 % Arguments for BoundSys
18 bound = 'U';
19 d = 4;                              % 2d = 8th degree auxiliary functional
20 epsilon = 0; sigma_noise = 0;       % No noise
21 verbose = 1;                        % Enable verbosity for SDP solver
22 symmetries = 0;                     % No simetries enable
23 SOSPsolver = 'yalmip';              % SOSP solver
24 SDPsolver = 'mosek';                % SDP solver
25 
26 % Magnitude to be bounded rescaled to the original value
27 magnitude = @(a) (rescaling^2)*(sigma*(a(1))^2 + a(2)^2 + beta*a(3)^2);
28 
29 U = zeros(1,length(rho));
30 res = zeros(1,length(rho));
31 for i = 1:length(rho)
32     % Create the 3-simensional system for Lorenz attractor
```

```
    f  = "data/LORENZinput"+"beta"+beta+"sigma"+sigma+"rho"+rho(i)+".mat";
    % Build the initial system if not done yet
    if not(isfile(f))  initLorenz(beta,sigma,rho(i),rescaling,f); end
    % Call BoundSys
    [U(i), res(i)]  = BoundSys(bound,f,magnitude,d,epsilon,sigma_noise,verbose,
        symmetries,SOSPsolver,SDPsolver);
end

%% Plot U and res
subplot(2,1,1)
plot(rho,U,"bo--");
grid on, xlabel("$\rho$","Interpreter","latex"), ylabel("Dissipation","Interpreter
    ","latex");
subplot(2,1,2)
plot(rho,res,"bo--");
grid on, xlabel("$\rho$","Interpreter","latex"), ylabel("Residual","Interpreter","
    latex");
```

Figure 2 shows the upper bounds and residual obtained after running `Lorenz.m`.

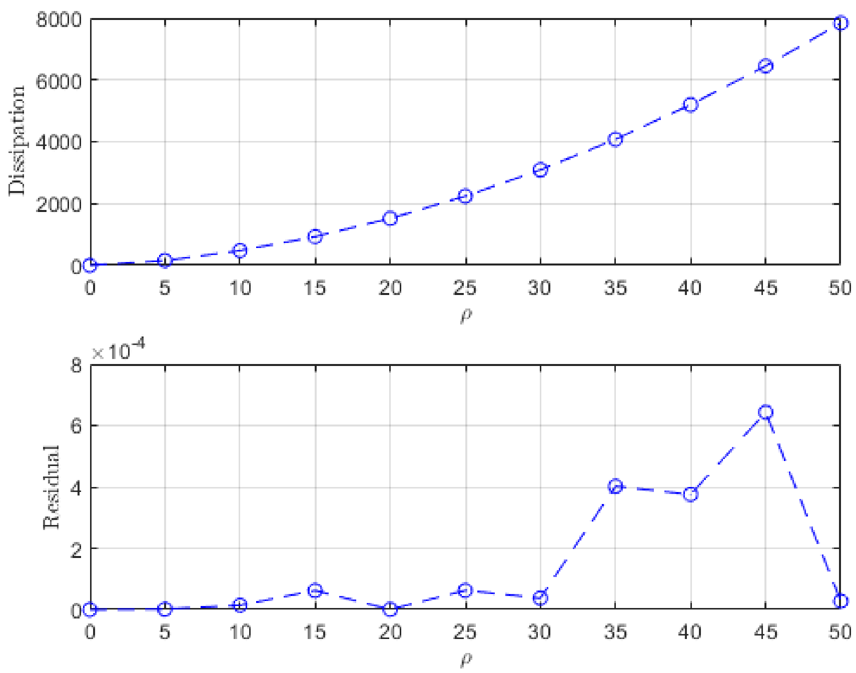

Figure 2: Upper bound on long time averaged dissipation in Lorenz attractor obtained after running `Lorenz.m`, and the corresponding residual.

3.4.2 example_KSE.m - N-dimensional Kuramoto-Sivashinsky Equation

Files `example_KSE.m` and `initKSE.m` located in `examples/KSE` are required to run this example.

In this example we look for a lower bound on steady mean energy in a 6-dimensional truncation of Kuramoto-Sivashinsky equations (KSE), to which noise with intensity $\epsilon = 10^{-3}$ is being added to all six components of the state vector $\mathbf{a}$. The code in `KSE.m` is included below. An analytical expression for the N-mode Galerkin truncation of KSE is [8]

$$f_i(\mathbf{a}) = \left(\frac{i}{\mathcal{L}}\right)^2 \left[1 - \left(\frac{i}{\mathcal{L}}\right)^2\right] a_i + \frac{1}{\sqrt{\pi \mathcal{L}}} \frac{i}{\mathcal{L}} \left[\frac{1}{2} \sum_{j=1}^{N-l} a_j a_{j+l} - \frac{1}{4} \sum_{j=1}^{l-1} a_j a_{l-j}\right] \tag{34}$$

for $i = 1, 2, ..., N$. $\mathcal{L}$ is a parameter. From this equation one can easily compute L_{ij} and N_{ijk}. We will consider $N = 6$ and $\mathcal{L} = 1.2$. This is done in line 28 by function `initKSE`, which also stores these tensors in the file indicated as third argument. A scaling factor equal to $2\pi L$ is applied to (34).

The expression for truncated mean energy reads

$$E(\mathbf{a}) = \frac{\mathbf{a}^T \mathbf{a}}{2\pi L} \tag{35}$$

For $\mathcal{L} < 1$ the solution $\mathbf{a} = 0$ is linearly unstable. This equilibrium point saturates the long time average and therefore, the lower bound obtained with $\epsilon = 0$ is 0. However, when adding noise with enough intensity, one expects that the all the possible trajectories go away from the unstable equilibrium at the origin.

From lines 17 to 23, the inputs to `BoundSys` are defined. Notice that the auxiliary function is of the 10^{th} degree, such high degrees are usually needed to deal with stochastic systems. The symmetry defined in line 23, with $\Lambda =$ `diag(symmetries)` is known to satify the conditions in §3.2.1. In this example, the inputs `SOSPsolver` and `SDPsolver` are not provided and will take their default value, i.e., `'yalmip'` and `'mosek'` respectively.

The output is

```
BoundSys:  Finding a bound for the 6-dimensional stochastic system...
BoundSys:  Using YALMIP.
BoundSys:  Valid symmetry.
------------------------------
Bound:  3.1942
Residual norm:  2.1641e-05
------------------------------
```

Listing 2: `KSE.m`

```
1 %%%%% KSE Finite-Dimensional System %%%%%
2 % Example of usage of initKSE(), UODESys() and BoundSys()
3 % to find a lower bound on long time avegared energy in
4 % Kuramoto-Sivashinsky equation with noise
5
6 % Written by Mario Lino Valencia (September 2019)
7 % Imperial College London - Department of Aeronautics
```

```
8
9  clear, clc;
10 mkdir data; % Create folder to store .mat with N_ijk, L_ij and B_i
11
12 N = 6;                         % Dimension of the N-dimensional truncated KSE
13 L = 1.2;                       % Length scale in KSE
14 rescaling = 2*pi*L;            % Rescaling factor
15
16 % Arguments for BoundSys
17 magnitude = @(a) (rescaling^2)*(a'*a)/(2*pi*L); % Magnitude to be bounded
18 bound = 'L';                   % Looking for lower bound
19 d = 5;                         % 2d = 10th degree auxiliary functional
20 epsilon = 1e-4/rescaling;      % Noise intensity
21 sigma    = eye(N);
22 verbose = 0;                   % Enable verbosity for SDP solver
23 symmetries = (-1).^(1:N);      % Simetries to enable block diagonalisation in SDP solver
24
25 %% Create the Finite-Dimensional System for KSE
26 f   = "data/KSEinputN" +N+"L"+L+".mat";
27 % Build the initial system if not done yet
28 if not(isfile(f))  initKSE(L,N,rescaling,f); end
29
30 %% Find an Upper Bound
31 [U,res,sol] = BoundSys(bound,f,magnitude,d,epsilon,sigma,verbose,symmetries);
32
33 disp("------------------------------");
34 disp("Bound:          " + U);
35 disp("Residual norm: " + res);
36 disp("------------------------------");
37
38 beep;
```

3.4.3 example_UKSE.m - n-dimensional uncertain Kuramoto-Sivashinsky Equation

Files `example_UKSE.m` and `initKSE.m` located in `examples/KSE` are required to run this example.

In this example we look for an upper bound on steady mean energy in a 6-dimensional truncation of Kuramoto-Sivashinsky equations (KSE) reduced to a 4-dimensional uncertain system. Noise with intensity $\epsilon = 10^{-4}$ is being added to the four most unstable components of the state vector $\mathbf{a}$, i.e., noise is added to the already reduced 4-dimensional system. The code in `UKSE.m` is included below.

An analytical expression for the N-mode Galerkin truncation of KSE is given by equation (34). We will consider $N = 6$ and $\mathcal{L} = 1.2$. From this equation one easily compute L_{ij} and N_{ijk}. This is done in line 27 by function `initKSE`, which also stores these tensors in the file indicated as third argument. A scaling factor equal to $2\pi L$ is applied to (34).

In this case, the expression for mean energy reads

$$E(\mathbf{a}, q^2) = \frac{\mathbf{a}^T\mathbf{a} + 2q^2}{2\pi L} \tag{36}$$

From lines 15 to 23, the inputs to `BoundUSys` are defined. Notice that the auxiliary function is of the 10^{th} degree, such high degrees are usually needed to deal with stochastic systems. In this example, the inputs `sigma`,`verbose`, `SOSPsolver` and `SDPsolver` are not provided and will take their default value, i.e., `eye(n)`,0, `'yalmip'` and `'mosek'` respectively. Since `sigma` is n-dimensional, noise is added to the already reduced system, what avoids possible stability issues.

The output obtained after running `UKSE.m` is

```
BoundUSys:  Finding a bound for the 4-dimensional uncertain system...
BoundUSys:  Using YALMIP
------------------------------
Bound:  3.255
Residual norm:  3.6243e-05
------------------------------
```

Listing 3: `UKSE.m`

```
1  %%%%% KSE Uncertain System %%%%%
2  % Example of usage of initKSE(), UODESys() and BoundUSys()
3  % to find an upper bound on energy in Kuramoto-Shivashinsky equation.
4
5  % Written by Mario Lino Valencia (September 2019)
6  % Imperial College London - Department of Aeronautics
7
8  clear, clc;
9  mkdir data;
10
11 N = 6;                        % Dimension of the N-dimensional truncated KSE
12 L = 1.2;                      % Length scale in KSE
13 rescaling = 2*pi*L;           % Rescaling factor
14
15 % Arguments for BoundSys
16 magnitude = @(a,q2) (rescaling^2)*(a'*a + 2*q2)/(2*pi*L); % Magnitude to be bounded
17 bound = 'U';                  % Looking for lower bound
18 d = 2;                        % 2d = 10th degree auxiliary functional
19 epsilon = 1e-4/rescaling;     % Noise intensity
20
21 % Inputs for UODESys
22 n = 4;                % Dimension of the uncertain system
23 g = [0.02,1,0.02];    % g = [g1 g2 g3] constants to determine the SoSP solved by
       UODESys
```

```
UODESys_file = "data/KSEoutputN"+N+"n"+n+"L"+L+".mat";

%% Create the Finite-Dimensional System for KSE
input_file  = "data/KSEinputN" +N+"L"+L+".mat";
if not(isfile(input_file))  initKSE(L,N,rescaling,input_file); end

%% Find an Upper Bound for the Uncertain System
[U,res,sol] = BoundUSys(bound,input_file,{n,g,UODESys_file},magnitude,d,epsilon);

disp("------------------------------");
disp("Bound:         " + U);
disp("Residual norm: " + res);
disp("------------------------------");

beep;
```

References

[1] Lakshmi M. Application of sum-of-squares of polynomials technique in fluid dynamics: Global stability, bounds for time-averages and flow control, 2017.

[2] Chernyshenko S I, Goulart P, Huang D, and Papachristodoulou A. Polynomial sum of squares in fluid dynamics: a review with a look ahead, 2014.

[3] Fantuzzi D, Goluskin D, Huang D, and Chernyshenko SI. Bounds for deterministic and stochastic dynamical systems using sum-of-squares optimization, 2016.

[4] Goluskin D and Fantuzzi G. Bounds on mean energy in the kuramoto–sivashinsky equation computed using semidefinite programming, 2019.

[5] Lofberg J. Pre- and post-processing sum-of-squares programs in practice, 2009.

[6] Lino M. Bounds on long-time averaged magnitudes in hydrodynamic-type systems using sum-of-squares of polynomials technique, 2019.

[7] Goulart PJ and Chernyshenko SI. Global stability analysis of fluid flows using sum-of-squares, 2012.

[8] Demetrios T. Papageorgiou and Yiorgos S. Smyrlis. The route to chaos for the kuramoto-sivashinsky equation. *Theoretical and Computational Fluid Dynamics*, 3(1):15–42, Sep 1991.