UODESys

A MATLAB toolbox for derivation of an uncertain finite-dimensional ODE system and finding related bounds

---

USER MANUAL

Author: **Mayur Venkatram Lakshmi**
Supervisor: Prof Sergei Ivanovich Chernyshenko

May 2017

Department of Aeronautics
Imperial College London
London SW7 2AZ
U.K.

1 Introduction

UODESys (**U**ncertain **O**rdinary **D**ifferential **E**quation **Sys**tem) is a free, third-party MATLAB® toolbox for derivation of an uncertain and reduced finite-dimensional system of ordinary differential equations (ODEs) from a system of ODEs of the following form:

$$\frac{da_i}{dt} = \sum_{j,k=1}^{N} N_{ijk} a_j a_k + \sum_{j=1}^{N} L_{ij} a_j + B_i \quad \text{for} \quad i = 1, ..., N, \tag{1}$$

where L_{ij} is an arbitrary two-dimensional matrix of size N x N, B_i is a vector with N elements and N_{ijk} is a three-dimensional array of size N x N x N, satisfying the condition:

$$\sum_{i,j,k=1}^{M} N_{ijk} a_i a_j a_k = 0 \; \forall \; M, \boldsymbol{a}. \tag{2}$$

In (2), $M \leq N$ and $\boldsymbol{a}$ is a vector of variables. An ODE system of the form (1) can be derived by applying a finite-dimensional Galerkin approximation to the incompressible Navier-Stokes equation, and is therefore of interest in the analysis of fluid flows.

We rewrite the first $n < N$ equations of (1) as:

$$\frac{da_i}{dt} = \sum_{j,k=1}^{n} N_{ijk} a_j a_k + \sum_{j=1}^{n} L_{ij} a_j + B_i + \Theta_i \quad \text{for} \quad i = 1, ..., n. \tag{3}$$

The terms Θ_i denote all the elements of the right hand side of (1) depending on $a_{n+1}, \ldots, a_N$. We can partition the vector of variables $\boldsymbol{a} = (a_1, ..., a_n, a_{n+1}, ..., a_N)$ into $\boldsymbol{u}$ and $\boldsymbol{v}$, defined as $(u_1, ..., u_n) = (a_1, ..., a_n)$ and $(v_1, ..., v_{N-n}) = (a_{n+1}, ..., a_N)$. Further, one can derive the following equation for dq^2/dt, where $q = \sqrt{||\boldsymbol{v}||^2/2}$ represents the energy of the residual variables $\boldsymbol{v}$:

$$\frac{dq^2}{dt} = -\boldsymbol{u} \cdot \boldsymbol{\Theta} + \Gamma + \chi + \sum_{i=n+1}^{N} B_i a_i. \tag{4}$$

This reduction was first proposed in [2]. For full details the reader is referred to [1].

A variable transformation can be performed which makes χ identically zero. This variable transformation can be performed for all the terms in (1). A new system of equations in the transformed variables $\hat{\boldsymbol{a}}$ can be derived, which, after carrying out the reduction procedure described above, results in the following:

$$\frac{d\hat{a}_x}{dt} = \sum_{y,z=1}^{n} \hat{N}_{xyz} \hat{a}_y \hat{a}_z + \sum_{y=1}^{n} \hat{L}_{xy} \hat{a}_y + \hat{B}_x + \hat{\Theta}_x \quad \text{for} \quad x = 1, ..., n \; , \tag{5}$$

$$\frac{d\hat{q}^2}{dt} = -\hat{\boldsymbol{u}} \cdot \hat{\boldsymbol{\Theta}} + \hat{\Gamma} + \sum_{x=n+1}^{N} \hat{B}_x \hat{a}_x. \tag{6}$$

Note that in (5) and (6), x, y and z represent integer values. Bounds are sought on $\hat{\Gamma}$ and $||\hat{\boldsymbol{\Theta}}||^2$. These bounds have the following form:

$$||\hat{\boldsymbol{\Theta}}||^2 \leq P\left(\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}\right), \tag{7a}$$

$$\hat{\Gamma} \leq \kappa \hat{q}^2, \tag{7b}$$

where P is a polynomial and κ is a constant. The system of equations (5) and (6), together with the bounds (7) constitutes an uncertain dynamical system.

2 What UODESys does

The developed software takes as inputs arbitrary arrays N_{ijk} (size N x N x N and satisfying condition (2)), L_{ij} (size N x N) and B_i (size N x 1). UODESys performs a variable transformation of the form $\boldsymbol{a} = T\hat{\boldsymbol{a}}$ which makes χ identically zero. The transformation matrix T is determined by setting up and solving an eigenvalue problem. This matrix T is then used to determine new arrays $\hat{N}_{xyz}$, $\hat{L}_{xy}$ and $\hat{B}_x$. These are the arrays N_{ijk}, L_{ij} and B_i respectively after coordinate transformation and truncation.

An upper polynomial bound for $||\hat{\boldsymbol{\Theta}}||^2$ for arbitrary inputs N_{ijk}, L_{ij} and B_i is found. The procedure for finding this bound relies on the techniques of sum-of-squares (SOS) optimization, which is explained in section 3. Additionally, a least upper bound for the $\hat{\Gamma}$ term for arbitrary input arrays is found using results from the eigenvalue problem that was previously set up in order to determine T.

The user is referred to [1] for the mathematical background and further details on how UODESys derives the uncertain dynamical system defined by (5), (6) and (7) for arbitrary inputs. Further, details on how the bounds are calculated are given in [1].

3 Sum-of-squares optimization

3.1 Sum-of-squares optimization programs

A SOS optimization program (SOSP) is an optimization problem of the following form:

$$\begin{aligned} &\min_{\boldsymbol{u}\in\mathbb{R}^m} [\boldsymbol{c}\cdot\boldsymbol{u}] \\ &\text{such that} \quad P_{i,0}(x) + P_{i,1}(x)u_1 + P_{i,2}(x)u_2 + ... + P_{i,m}(x)u_m \in \Sigma_m \quad \text{for} \quad i = 1, ..., k, \end{aligned} \tag{8}$$

where Σ_m denotes the set of all sum-of-squares polynomials in $\mathbb{R}^m$. The vector of weighting coefficients $\boldsymbol{c}$ and the polynomials $P_{i,j}$ ($i = 1, ..., k$ and $j = 1, ..., m$) are to be specified as part of the problem definition. The elements of the vector $\boldsymbol{u} \in \mathbb{R}^m$ are referred to as the decision variables. SOS optimization problems of the form (8) can be solved efficiently by reformulating them as a semidefinite program (SDP) [4].

UODESys uses the MATLAB toolbox YALMIP [5] to automate the conversion from SOSP to SDP. YALMIP then calls the SDP solver SEDUMI [7], and finally YALMIP converts the SDP solution back to the solution of the original SOSP. As detailed in [6], the steps required to define and solve a SOSP are as follows:

- Declare the SOSP decision variables
- Define the SOSP constraints
- Set objective function
- Call SDP solver
- Get solutions

A SOSP is set up and solved to find the polynomial bound for $||\hat{\boldsymbol{\Theta}}||^2$.

3.2 Finding a bound for $||\hat{\boldsymbol{\Theta}}||^2$

Using the SOS methods detailed in section 3.1, a polynomial upper bound P for $||\hat{\boldsymbol{\Theta}}||^2$ is sought in the form:

$$||\hat{\boldsymbol{\Theta}}||^2 \leq P(\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}) = c_1 \frac{||\hat{\boldsymbol{v}}||^2}{2} + c_2||\hat{\boldsymbol{u}}||^2||\hat{\boldsymbol{v}}||^2 + c_3||\hat{\boldsymbol{v}}||^4, \tag{9}$$

where $||\ .\ ||$ represents the Euclidean norm, ie: $||\hat{\boldsymbol{u}}||^2 = \hat{u}_1{}^2 + \ldots + \hat{u}_n{}^2$ and $||\hat{\boldsymbol{v}}||^2 = \hat{v}_1{}^2 + \ldots + \hat{v}_{N-n}^2$. A full proof of the existence of the coefficients c_1, c_2 and c_3 can be found in Appendix A of [2].

To obtain an optimal upper bound, a linear combination of the decision variables $\gamma_1 c_1 + \gamma_2 c_2 + \gamma_3 c_3$ is minimised subject to the constraint that $P - ||\hat{\boldsymbol{\Theta}}||^2$ is positive semidefinite. This positive semidefinite constraint is recast as a sum-of-squares constraint, leading to the following optimization problem:

$$\begin{aligned} &\min_{c_1,c_2,c_3} \ [\gamma_1 c_1 + \gamma_2 c_2 + \gamma_3 c_3] \\ &\text{such that} \quad P\,(c_1, c_2, c_3, \hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}) - ||\hat{\boldsymbol{\Theta}}||^2 \in \Sigma_n, \end{aligned} \tag{10}$$

where Σ_n denotes the set of all sum-of-squares polynomials in $\mathbb{R}^n$. The values of γ_1, γ_2 and γ_3 have to be specified as inputs to the code, and UODESys prompts the user for these values. The optimization problem (10) is solved by defining the sum-of-squares constraint and the objective function in YALMIP syntax. When the code is run, YALMIP then invokes the SDP solver SEDUMI. The obtained optimal values of c_1, c_2 and c_3 are substituted into (9) to determine the polynomial bound P for $||\hat{\boldsymbol{\Theta}}||^2$.

4 Requirements and Installation

In order for MATLAB to execute the script files comprising UODESys, the user must install the MATLAB toolbox YALMIP by downloading the set of `.m` files that constitute YALMIP and adding these files to MATLAB's working directory. A download link to the latest release of YALMIP, as well as installation instructions are available at the following URL: `https://yalmip.github.io`. Note that MATLAB 5.2 or earlier versions are not supported by YALMIP.

In addition, it is necessary to install an SDP solver such as SEDUMI. A download link for SEDUMI can be found at: `http://sedumi.ie.lehigh.edu`. To install SEDUMI, the user should download the `.zip` archive containing SEDUMI's constituent files, unzip it, and add the resulting folder to the MATLAB working directory.

UODESys consists of five MATLAB `.m` files:
INPUT.m, **Variable_Transformation.m**, **Bounds_Calculation.m**,
Random_Array_Generator.m, **Validation.m**.

All of these files should also be downloaded and added to MATLAB's working directory. The first three of these files are essential components of UODESys and are described in the next section. Random_Array_Generator.m and Validation.m are two additional tools that are introduced in section 7. To verify that all components have been correctly installed, refer to the illustrative example in section 6 of this manual. By defining matrices `N_ijk`, `L_ij` and `B_i` as given and running the UODESys scripts, one should obtain outputs that are identical to those presented in section 6.

5 Instructions to run software

5.1 INPUT.m - STEP 1

The user should run the MATLAB script **INPUT.m** to read in matrices N_{ijk}, L_{ij} and B_i from disk. N_{ijk} should be stored as a 3-dimensional array with the variable name `N_ijk`, L_{ij} should be stored as a 2-dimensional matrix with the variable name `L_ij`, and B_i should be stored as a column vector with the variable name `B_i`. In all these variables, the index `i` represents the first dimension in the array and corresponds to the index i in equation (1). The index `j` represents the second dimension in the arrays N_{ijk} and L_{ij}. The index `k` represents the third dimension in the array N_{ijk}. Note that, in accordance with standard convention, the first index (`i`)

indicates row number, the second index (`j`) indicates column number and the third index (`k`) refers to the third dimension (ie: indicates layer number).

All three of these variables should be stored in a MATLAB formatted binary file (MAT-file) called `INPUT.mat`. This `.mat` file must be saved in the directory in which the `.m` files comprising UODESys are stored. The script INPUT.m checks that $\sum_{i,j,k=1}^{M} N_{ijk}a_i a_j a_k = 0 \; \forall \; M \leq N$ and random $\boldsymbol{a}$. If this condition is not satisfied for random $\boldsymbol{a}$ and all $M \leq N$, a message is displayed in the command window warning the user that $N_{ijk}a_i a_j a_k \neq 0$. To perform this test, a vector of randomly generated a_i $(i = 1, ..., N)$ values is created. The input array `N_ijk` is analysed in this script to determine the value of N, which is the size of the original system of ODEs.

5.2 Variable_Transformation.m - STEP 2

Next, the script **Variable_Transformation.m** should be run. This script uses the arrays `N_ijk`, `L_ij` and `B_i` obtained from step 1. The L_{ij} matrix is expressed as a symmetric matrix $\frac{1}{2}\left(L_{ij} + L_{ij}^T\right)$, and the matrix A is defined to be $2\left(L_{ij} + L_{ij}^T\right)$. The script calculates the eigenvalues and eigenvectors of the A matrix. The eigenvalues λ_i are sorted in descending order with λ_1 being the largest. The eigenvectors are also correspondingly ordered and stored in a matrix variable with the name `T`. Each column of this matrix is an eigenvector of A. Using this matrix of eigenvectors, the script calculates new arrays $\hat{N}_{xyz}$, $\hat{L}_{xy}$ and $\hat{B}_x$, which are the arrays N_{ijk}, L_{ij} and B_i respectively in the new (transformed) variables. These new arrays are stored in variables named `N_hat_xyz`, `L_hat_xy` and `B_hat_x` respectively. A file named `transformed_arrays.mat` is created. The transformed arrays, cell array representations of these arrays, as well as other workspace variables required for the next step are saved in this file.

5.3 Bounds_Calculation.m - STEP 3

Finally, the script **Bounds_Calculation.m** is to be run, which reads in variables from the MATLAB binary file `transformed_arrays.mat` produced when running Variable_Transformation.m in step 2. After this is done, the user is prompted in the MATLAB command window to input the value of n (the size of the reduced system of ODEs), as well as the values of γ_1, γ_2 and γ_3, which are required to solve the optimization problem described in section 3.2 of this manual. Note that UODESys has been designed to work with values of n that are strictly less than N. If a value of n is entered that is equal to or greater than N, Bounds_Calculation.m will not run correctly and an error message will appear in the MATLAB command window.

The user should refer to the diagonal entries in the `eigenvalues` matrix in order to choose a suitable value for n. The value κ, used to determine the bound for $\hat{\Gamma}$, is defined to be $-\lambda_{n+1}/2$. Hence, a bound for $\hat{\Gamma}$ is found. A bound for $||\hat{\boldsymbol{\Theta}}||^2$ is then determined using SOS methods. The final output is stored in a new file `OUTPUT.mat` which is saved to disk. Among other outputs, this file also contains a matrix with variable name `L_hat_xy_star`. This is the coefficient matrix for the χ term in the new (transformed) coordinates. The user can verify that all elements of this matrix are zero (to within rounding error). Note that `L_hat_xy_star` will be a matrix of zeros only if the condition (2) is satisfied. The contents of the output file `OUTPUT.mat` is detailed in the following table:

Output Variable	Description
N_hat_xyz_final	3D Array N_{ijk} after variable transformation and truncation
L_hat_xy_final	2D Matrix L_{ij} after variable transformation and truncation
B_hat_x_final	Column vector B_i after variable transformation and truncation
L_hat_xy_prime	Coefficient matrix for Γ (after variable transformation)
L_hat_xy_star	Coefficient matrix for χ (after variable transformation)
KAPPA	$\hat{\Gamma} \leq \kappa \hat{q}^2$
c_1_solution	Coefficient of $\|\|\hat{\boldsymbol{v}}\|\|^2/2$ in polynomial bound P
c_2_solution	Coefficient of $\|\|\hat{\boldsymbol{u}}\|\|^2\|\|\hat{\boldsymbol{v}}\|\|^2$ in polynomial bound P
c_3_solution	Coefficient of $\|\|\hat{\boldsymbol{v}}\|\|^4$ in polynomial bound P
eigenvalues	Diagonal matrix of eigenvalues sorted in descending order
eigenvectors	Matrix of eigenvectors used in coordinate transformation (T = eigenvectors)
Q_eigenvalues	Vector containing eigenvalues of Q, where $P - \|\|\hat{\boldsymbol{\Theta}}\|\|^2$ is decomposed as $\boldsymbol{Z}^T Q \boldsymbol{Z}$
N_hat_xyz	3D Array N_{ijk} after variable transformation and before truncation
L_hat_xy	2D Matrix L_{ij} after variable transformation and before truncation
B_hat_x	Column vector B_i after variable transformation and before truncation

6 Illustrative Example

This section details an example of running the codes. The value of N was fixed to be 4, and arrays N_{ijk}, L_{ij} and B_i of sizes 4 x 4 x 4, 4 x 4 and 4 x 1 respectively were generated. These arrays were stored in a file INPUT.mat and are as follows:

```
N_ijk(:,:,1) =
         0         0         0         0
   -0.0366   -0.1771   -0.0465   -0.6454
   -0.0872   -0.8595   -0.8368   -0.3628
   -0.9828   -0.0337   -0.0219   -0.1989

N_ijk(:,:,2) =
    0.0366    0.1771    0.0465    0.6454
         0         0         0         0
   -0.1714   -0.3346   -0.6588   -0.7579
   -0.2357   -0.6318   -0.0888   -0.9766

N_ijk(:,:,3) =
    0.0872    0.8595    0.8368    0.3628
    0.1714    0.3346    0.6588    0.7579
         0         0         0         0
   -0.5981   -0.8129   -0.1867   -0.0769

N_ijk(:,:,4) =
    0.9828    0.0337    0.0219    0.1989
    0.2357    0.6318    0.0888    0.9766
    0.5981    0.8129    0.1867    0.0769
         0         0         0         0
```

```
L_ij =
    0.1405    0.6498    0.2968    0.0241
    0.3894    0.4911    0.1985    0.3941
    0.7121    0.9685    0.2192    0.2469
    0.0326    0.9337    0.9805    0.0597

B_i =
    0.2787
    0.0885
    0.0320
    0.9279
```

The above arrays are read in from file using the script **INPUT.m**. When INPUT.m is run, the following message is displayed in the MATLAB command window informing the user that the condition $\sum_{i,j,k=1}^{M} N_{ijk} a_i a_j a_k = 0$ is satisfied ($\forall\ M \leq N$ and random $\boldsymbol{a}$):

```
>> INPUT
N_ijk * a_i * a_j * a_k = 0 (Input N_ijk is okay)
```

The script **Variable_Transformation.m** determines the eigenvectors and eigenvalues of the matrix $2\left(L_{ij} + L_{ji}\right)$. The eigenvalues are sorted in descending order and stored in a diagonal matrix and the eigenvectors are also correspondingly ordered. The resulting matrices are displayed in the command window when this script is run:

```
>> Variable_Transformation
EIGENVECTORS of 2(L_ij + L_ji)
   −0.3713    0.8000    0.0280    0.4705
   −0.6228   −0.0678   −0.7041   −0.3343
   −0.5303    0.0131    0.6970   −0.4825
   −0.4393   −0.5960    0.1331    0.6589

EIGENVALUES of 2(L_ij + L_ji)
    7.0640    0.0000   −0.0000    0.0000
    0.0000    0.3345   −0.0000   −0.0000
   −0.0000   −0.0000   −0.9310   −0.0000
    0.0000   −0.0000   −0.0000   −2.8251
```

This matrix of eigenvectors is used to perform a variable transformation, after which, new arrays $\hat{N}_{xyz}$, $\hat{L}_{xy}$ and $\hat{B}_x$ are obtained.

When running the next script **Bounds_Calculation.m**, the user is prompted to enter the value of n, the size of the truncated system. The user is also prompted to enter the values of γ_1, γ_2 and γ_3. The user should consult the diagonal matrix of eigenvalues to choose a suitable value of n. To guarantee stability, n is chosen so that λ_{n+1} is negative. In this case, n is chosen to be equal to 2, and γ_1, γ_2 and γ_3 are each set to be equal to 1. The following is displayed in the command window when running Bounds_Calculation.m with these values:

```
>> Bounds_Calculation
Enter value of n, the size of the reduced system of ODEs (n < N): 2
Enter value of gamma_1 (Weighting coefficient of c_1): 1
Enter value of gamma_2 (Weighting coefficient of c_2): 1
Enter value of gamma_3 (Weighting coefficient of c_3): 1
```

Immediately after this, the transformed arrays $\hat{N}_{xyz}$, $\hat{L}_{xy}$ and $\hat{B}_x$ are displayed as follows:

```
N_hat_xyz_final (N_ijk after variable transformation and truncation):
(:,:,1) =
   −0.0000    0.0000
    1.5568   −0.0962

(:,:,2) =
   −1.5568    0.0962
    0.0000   −0.0000

L_hat_xy_final (L_ij after variable transformation and truncation):
    1.7660   −0.2146
    0.2146    0.0836

B_hat_x_final (B_i after variable transformation and truncation):
   −0.5832
   −0.3357.
```

Note that the above displayed arrays are the arrays in equation (5) of this manual. The coefficient matrix for the $\hat{\Gamma}$ term is then displayed, together with the values of λ_{n+1} and κ.

```
L_hat_xy_prime (The coefficient matrix of GAMMA_hat):
   −0.2328    0.0000
    0.0000   −0.7063
(n+1)th eigenvalue of A = 2*(L_ij + L_ji):
   −0.9310

KAPPA = :
   −0.4655
```

Note that $\hat{\Gamma} \leq \kappa \hat{q}^2$ and κ is defined to be equal to be $-\lambda_{n+1}/2$. SOS methods are then used to calculate an upper bound for $||\hat{\mathbf{\Theta}}||^2$. Specifically, the optimization problem described in section 3.2 is set up and solved to find the values of c_1, c_2 and c_3. YALMIP and SEDUMI then display in the command window information relating to the solution of the optimization problem. This is followed immediately by the calculated optimal values of c_1, c_2 and c_3:

```
c1:
    1.3896

c2:
    2.4497

c3:
    0.4478.
```

The above displayed variables constitute the definition of an uncertain dynamical system in the form of equations (5), (6) and (7) for the given inputs N_{ijk}, L_{ij} and B_i. The user can verify that the matrix with the variable name `L_hat_xy_star` is a matrix of zeros. Indeed, this should be case, as the χ term after variable transformation and reduction of the ODE system is identically zero, and the matrix `L_hat_xy_star` is the coefficient matrix for this term. These aforementioned variables, together with other workspace variables are saved to disk in a file `OUTPUT.mat`.

7 Additional Tools

The user is referred to [1] for additional details on the tools presented below.

7.1 Random_Array_Generator.m

The MATLAB function **Random_Array_Generator.m** can be used to generate random input arrays. The function takes as an input the value of N. The first output is a random 3D array N_{ijk} (of size N x N x N) which satisfies the condition $N_{ijk} = -N_{kji}$, which is a sufficient condition for $\sum_{i,j,k=1}^{M} N_{ijk} a_i a_j a_k = 0 \; \forall \; M \leq N, \boldsymbol{a}$. Random L_{ij} and B_i are also returned, together with the value of N. This MATLAB function could be used for the purposes of testing and experimentation. This function should be called in the following format:

```
>> [N_ijk, L_ij, B_i, N] = Random_Array_Generator(N)
```

7.2 Validation.m

The MATLAB script **Validation.m** is used to validate the output and assess the quality of the obtained bounds. It is checked whether the condition (2) is satisfied in the transformed coordinates. The correctness of the variable transformation is also checked with another test, and three plots are produced to aid the user in assessing the quality of the bounds.

References

[1] Lakshmi M. *Application of sum-of-squares of polynomials technique in fluid dynamics*. MEng Thesis. Imperial College London; 2017.

[2] Goulart PJ, Chernyshenko S. Global stability analysis of fluid flows using sum-of-squares. *Physica D: Nonlinear Phenomena*. 2012;241(6): 692-704.

[3] Chernyshenko SI, Goulart P, Huang D, Papachristodoulou A. Polynomial sum of squares in fluid dynamics: a review with a look ahead. *Philosophical Transactions of the Royal Society A (Mathematical, Physical & Engineering Sciences)*. 2014;372(2020): 20130350 (18 pp.).

[4] Papachristodoulou A, Prajna S. A tutorial on sum of squares techniques for systems analysis. *Proceedings of the 2005 American Control Conference, 8-10 June 2005*. Piscataway, NJ, USA: IEEE; 2005. pp. 2686-700.

[5] J. Lofberg. YALMIP : a toolbox for modeling and optimization in MATLAB. *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, ; 2004. pp. 284-289.

[6] Papachristodoulou A, Anderson J, Valmorbida G, Prajna S, Seiler P, Parrilo P. SOSTOOLS Version 3.00 Sum of Squares Optimization Toolbox for MATLAB. *ArXiv e-prints*. 2013. Available from: http://sysos.eng.ox.ac.uk/sostools/.

[7] Sturm JF. Using SeDuMi 1.02, A Matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*. 1999;11(1-4): 625-653.