README: Matlab Code Suite for Optimal Least Action Control

This Matlab code suite is intended to accompany the publication "Control of stochastic and induced switching in biophysical complex networks", and constitutes an implementation of the algorithm given there, Optimal Least Action Control (OLAC), as well as a worked example. Its purpose is to be a tool for those looking to use OLAC in their own research to get a working version of the algorithm up and going quickly.

To run this code with the given example, the script `OptimalLeastActionControl_Top.m` can be executed in the Matlab shell "off-the-shelf", without any modifications. The script calls the main function, `OptimalLeastActionControl.m` on a pre-built example. Before reading further, it is suggested that this be tried.

OLAC has six required functions/data files. Furthermore, parameters related to the continuation and optimization procedures can be modified from the given values. This file discusses what each of those functions/data sets are, and discusses the parameters involved through the example that the included script acts on.

Required User Defined Functions

1. The first required file is a **vectorized dynamical system**. In our example, we apply OLAC to a two dimensional dynamical system of mutual inhibition and self excitation (MISE), given as

$$\begin{aligned}\frac{dx_1}{dt} &= \frac{x_1^4}{p_1^4 + x_1^4} + \frac{p_2^4}{p_2^4 + x_2^4} - x_1,\\ \frac{dx_2}{dt} &= \frac{x_2^4}{p_3^4 + x_2^4} + \frac{p_4^4}{p_4^4 + x_1^4} - x_2,\end{aligned} \tag{1}$$

where the p_i, $i = 1, \ldots, 4$ are the four tunable parameters for this system. This function should be written to take in three variables, in this order:

   (a) time (t),

   (b) state ($\vec{X}$),

   (c) parameters ($\vec{p}$).

   The vectorized Matlab implementation of this system is found in the file `MutualInhibitionSelfExcitation.m`.

2. The second required file is an **array of stable states of the system**, given as an $N \times M$ dimensional array, where N is the dimension of the system (in this case $N = 2$), and M is the number of stable states (denoted $\vec{X}_i^*$) under consideration. In the case of the example, $M = 3$:

$$\vec{X}_1^* = \begin{pmatrix} 1.996 \\ 0.004 \end{pmatrix}, \quad \vec{X}_2^* = \begin{pmatrix} 0.004 \\ 1.996 \end{pmatrix}, \quad \vec{X}_3^* = \begin{pmatrix} 1.00 \\ 1.00 \end{pmatrix}. \tag{2}$$

   These stable states are stored in the file `MISE_StableStates.mat`.

3. The third required function is the **OLAC objective functional**: the functional that evaluates the complete array of actions associated with transition between stable states and returns a single value for how close those actions meet a desired goal. If it is expected that stable states will be lost through bifurcation in the course of the optimization process, the functional should account for this. It should take in two variables, in this order:

   (a) The array of action values for all possible transitions, A. Note that this array is calculated in the code such that all indirect transitions are given a value of `Inf`. Furthermore, only those states which have not been lost to bifurcation are included.

   (b) A list of which states the original specified stable fixed points have been modified to through bifurcation (found using the stable state classification function discussed below).

   In the example, our OLAC objective functional converts the action matrix to a rate matrix through the formula $\exp\left(-\frac{S^*}{\varepsilon}\right)$, and returns the negative of the occupancy of the third state, $\vec{X}_3^*$.

   This functional is given as the Matlab function `MISE_OLACFunctional.m`.

4. The fourth required file is a **stable state classification function**. At each progressive parameter set used in OLAC, the algorithm numerically continues the stable fixed points from their original positions corresponding to the original parameters to their new positions corresponding to the current parameter set. In doing so bifurcations may occur, resulting in a particular stable state becoming unstable. This in turn could affect the behavior of the objective function. To account for this, a stable state classification function should be included that can classify which original stable state a given state is a continuation of. Such a function should take in a vector $\vec{X}^*$, a stable state of the system, and should return an integer between 1 and M corresponding to which of the stable fixed points $\vec{X}^*$ is. An example of such a stable state classification function used in the present application of OLAC based on calculating the nearest original fixed point to a given state can be found in `MISE_SSC.m`. Note that this function may not be required, depending on the application.

5. The fifth required file is a **vectorized analytic Jacobian of the dynamical system**. While not strictly required, providing an analytic Jacobian for the dynamical system will speed up the computation of transition paths by up to 100-fold. It is the single most substantial thing that can be done to increase the performance of OLAC. Like the dynamical system function, the Jacobian function should take in three variables:

   (a) time (t),

   (b) state ($\vec{X}$),

   (c) parameters ($\vec{p}$).

   An example vectorized Jacobian is given in `MISE_Jacobian.m`. Constants and/or zeros in the Jacobian can be vectorized in the proper way by changing them, respectively, to `CONSTANT*ones(1, 1, tsteps)` or `zeros(1, 1, tsteps)`.

6. The sixth and final required file is a **bifurcation constraint function** on the parameters of the system, $\vec{p}$. Such a function should take in these parameters (the current parameter state of the system), and return the largest real eigenvalue part of each stable state (or a specified, nonempty subset of the stable states) as these states are numerically continued from the original, specified parameter set to the current one. Note that other inequality and

equality constraints on the parameters (e.g., a sparsity constraint) can be specified as well in accordance with the Matlab function `fmincon` documentation. Further information on nonlinear constraints in `fmincon` can be found here:
`http://www.mathworks.com/help/optim/ug/writing-constraints.html#brhkghv-16`.

An example bifurcation constraint function is given in `MISE_Constraint.m`. This function takes in a single variable, $\vec{p}$, the current parameters of the system.

OLAC Parameters

OLAC requires that 6 parameters be set by the user. The top level script `OptimalLeastActionControl_Top.m` gives documentation on each of these, along with good baseline values. In particular, the bounds on the objective parameters can be modified to help improve convergence of fmincon. Other internal parameters in the function `OptimalLeastActionControl.m` and `MISE_Constraint.m` can be modified as well.

Output

The Matlab `struct` returned by OLAC has three fields:

1. `ObjVal`: the value of the optimized objective function.
2. `Params`: the set of parameters on which the objective function achieves its optimal value.
3. `History`: the history of the optimization run by OLAC.