Dissipativity-Augmented H-infinity Control

This project designs an H-infinity controller satisfying a prescribed QSR-dissipativity constraint. The code is based on Algorithm 1[[1]](#footnote-1) from the manuscript “Dissipativity-Based Robust Control with H-infinity Optimal Performance” by Ethan J LoCicero and Leila Bridgeman, which is currently under review for presentation at IFAC World Congress 2023.

Use this project as follows.

1. Run *plant.m*. This randomly generates an LTI system for which the controller will be designed and stores the data in *plantdata.mat*. Alternatively, a different system of interest could be constructed and stored in *plantdata.mat*, as long as its dimensions and matrices are assigned as described in *plant.m* and the plant is stable.
2. Run *initialize.m*. This loads *plantdata.*mat, identifies a QSR dissipative description of the plant, generates an initial feasible point for Algorithm 1, and stores the data in *initialization.mat*.
3. Run *ICO.m*. This loads *initialization.mat* and implements Algorithm 1 (iterative convex overbounding) to iteratively minimize the closed-loop H-infinity norm while satisfying the prescribed dissipativity constraint. At each iteration, the iteration number and the change in the closed-loop H-infinity norm is printed. Once Algorithm 1 converges to a local minimum, several check are made to ensure the solution is accurate. The printed outputs are as follows:
   1. *plantIsQSR* checks that the plant is in the identified dissipative bounds. 1 true, 0 is false.
   2. *controllerIsQSR* checks that the controller is in the prescribed dissipative bounds. 1 is true, 0 is false.
   3. *isStable* checks that the plant and controller QSR bounds imply input-output stability through the Dissipative Theorem. 1 is true, 0 is false.
   4. *isGammaBounded\_new* checks that the closed-loop satisfies the stated H-infinity norm bound. 1 is true, 0 is false.
   5. *percent\_improvement* displays the percent decrease in the closed-loop H-infinity norm from the initial controller. This value should be between 0 and 100.

The resulting controller parameters (*Ac,Bc,Cc,Dc)*, closed-loop H-infinity norm (*gamma(k))*, and all auxiliary data—such as the total calculation time (*time\_calc)—*is stored in *data.mat*.

These scripts require proper installations of *yalmip* and *mosek* to solve semidefinite programs.

Furthermore, these scripts call several other functions:

* *findHinf.m* returns the H-infinity norm of the input system, which must be stable LTI
* *HinfCD.m* adjusts the output matrices of a given system to satisfy a desired H-infinity norm
* *findQSRboth.m* identifies complementary QSR descriptions of two systems that satisfy the Dissipativity Theorem
* *HinfQSRsimple.m* solves the main optimization problem in each step of Algorithm 1 in order to minimize the closed-loop H-infinity norm.
* *UpdateL\_.m and UpdateLambda\_.m* solve optimization problems to initialize or update weighting matrices between iterations.
* *checkQSR.m* determines whether an LTI system satisfies certain QSR-dissipativity bounds
* *checkQSRtheorem.m* determines whether two sets of QSR-dissipative bounds satisfy the Dissipativity Theorem
* *checkHinf.m* determines whether an LTI system satisfies an H-infinity norm bound

1. There are two important differences between this code and Algorithm 1. First, the initialization procedure does not use QSRcd (i.e. Algorithm 2). Instead, it uses a simpler procedure to arrive at an initial feasible point, which can only handle stable plants. Second, Qc is not required to be negative definite. This results in some changes to the QSR LMI from Theorem 5. These differences exist because this project is a more basic proof of concept for the iterative convex overbounding approach than the final product presented in the manuscript. As such, this project cannot be used to replicate exact results from the paper. [↑](#footnote-ref-1)