edX

Course › Week 4 › Option... › OPTIO...

OPTIONAL:Barrier Functions

Barrier Functions (Part A)

0.0/5.0 points (ungraded)

Prerequisites for this problem: You will need a semidefinite program solver for this problem. If you haven't done so already, please install either SeDuMi (found here) or MOSEK (found here). MOSEK is typically a lot faster than SeDuMi and we recommend installing it.

In class we saw how to use Lyapunov functions to prove regional stability of systems. However, in many robotics applications we are not directly interested in proving stability of the system to a fixed point. Rather, we care more about our system avoiding unsafe regions such as obstacles in the environment. In this problem, we will explore the notion of "barrier functions", which can be used to prove safety of dynamical systems.

More precisely, suppose we have a system described by the equation:

$$\dot{x} = f(x),$$

along with a set X_0 of initial conditions the system could start off in and an "unsafe region" X_u that the system must avoid. We would like to guarantee that any trajectory starting in the set X_0 avoids the region X_u. A sufficient condition for this is to find a *barrier function* $B(x)$ satisfying the following conditions:

$$\begin{aligned} &B(x) < 0,\ \forall x \in X_0 \ \ (C1) \\ &B(x) \geq 0,\ \forall x \in X_u \ \ (C2) \\ &\dot{B}(x) = \frac{\partial B(x)}{\partial x} f(x) \leq 0,\ \forall x \ \ (C3)\,. \end{aligned}$$

It is straight-forward to see that these conditions guarantee that trajectories starting in X_0 do not enter X_u. This is because for any trajectory starting in X_0, the value of the barrier function at time 0 is negative. Since the time-derivative of $B(x)$ is non-positive everywhere, the value of $B(x)$ cannot increase and thus remains negative. Since $B(x)$ is nonnegative inside the unsafe region, this is a proof that trajectories don't enter it! In fact we have proven that trajectories cannot cross the "barrier" formed by the 0 level-set of $B(x)$.

In this problem we will walk you through the process of using Sums-of-Squares (SOS) Programming to search for barrier functions. We will consider the following system:

$$\dot{x}_1 = x_2,$$
$$\dot{x}_2 = -x_1 + x_1^3 - x_2.$$

The initial condition set is the disc of radius 0.5 centered around the point (1.5,0). The unsafe set is the disc of radius 0.4 centered around the point (-1,-1).

(a) Write down a **quadratic polynomial** g_{X_0} that is **nonnegative inside the initial condition set and negative outside it**.

```
syms x1 x2
g_X0 = ;

```

Unanswered

```
syms x1 x2
% One such polynomial is:
g_X0 = 0.5^2 - (x1 - 1.5)^2 - x2^2;
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Barrier Functions (Part B)

0.0/5.0 points (ungraded)

(b) Write down a **quadratic polynomial** g_{X_u} that is **nonnegative inside the unsafe set and negative outside it**.

```
syms x1 x2
g_Xu = ;

```

Unanswered

```
syms x1 x2
% One such polynomial is:
g_Xu = 0.4^2 - (x1+1)^2 - (x2+1)^2;
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Barrier Functions (Part C)

0.0/15.0 points (ungraded)

(c) We have provided some stub code (found here) that uses the Systems Polynomial Optimization Toolbox (SPOT) to setup SOS constraints for our problem (this toolbox comes with Drake). You only need to fill out the lines in the blocks of code that say "FILL ME IN".

First, copy and paste your solutions to parts (a) and (b) of the problem in order to define g_{X_0} and g_{X_u}. Next, fill in the SOS constraints corresponding to the constraints (C2) and (C3) above. Constraint (C1) has already been implemented for you in the code.

Assuming these have been implemented correctly, you should see a plot with the initial condition set, unsafe set, the vector field, and the 0 level-set of the barrier function. An easy way to check that things are running correctly is to check the following in the plot:

- The 0 level-set of the barrier function (plotted in blue) separates the initial set and the unsafe set.

- The vector field points "inwards" (i.e., towards the side that contains the initial condition set) along the 0 level-set of the barrier function.

Once you are satisfied that the code is running correctly, type in the barrier function below (this is printed for you to the MATLAB screen). Note: You do not have to modify V in any way (e.g., taking out parentheses, etc. or changing the names of variables). Simply copy it below.

```
syms x1 x2
B = ;

```

Unanswered

```
% The code with everything filled in (and some comments) is provided below

% Initialize SOS program
checkDependency('spotless');
prog = spotsosprog;

ok_mosek = checkDependency('mosek');
ok_sedumi = checkDependency('sedumi');

if ~ok_sedumi && ~ok_mosek
        error('You need either MOSEK or SeDuMi installed to use this function.');
end

% Choose sdp solver
if ok_mosek
    solver = @spot_mosek;
    solver_name = 'mosek';
else
    solver = @spot_sedumi;
    solver_name = 'sedumi';
end

% solver = @spot_sedumi;



% State (x = [x1;x2])
x1 = msspoly('x');
x2 = msspoly('y');
x = [x1;x2];
prog = prog.withIndeterminate(x);

% Dynamics
f = [x2; -x1 + x1^3 - x2];

% Initialize barrier function as a free quartic polynomial
d = 4;
[prog,B] = prog.newFreePoly(monomials(x,0:d));

% Compute time derivative of B
Bdot = diff(B,x)*f;


%%FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description of initial condition set (copy and paste your answer from
% part (a) of the problem)
g_X0 = 0.5^2 - (x1 - 1.5)^2 - x2^2; % Should be in terms of x1 and x2
```

```
% Description of unsafe set (copy and paste your answer from
% part (b) of the problem)
g_Xu = 0.4^2 - (x1+1)^2 - (x2+1)^2; % Should be in terms of x1 and x2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% SOS constraints
% C1: B(x) < 0 inside initial condition set
% Previously in the problem set, we defined a function g_X0 that is nonnegative
% inside X0 and negative outside it. Then, the sums-of-squares (SOS) constraint
% we want is:
%
% p(x) = -B(x) - L(x)g_X0(x) is SOS, (1),
% L(x)$ is SOS, (2).
%
% For convenience, we'll denote q(x) = -L(x)g_X0(x). Now, if we have a point x0 in
% X0, q(x0) is negative (since L(x) is nonnegative and g_X0 is nonnegative in X0).
% Hence, for p(x) to be nonnegative at x0 (which it must be since we've constrained
% it to be SOS), -B(x0) must be nonnegative. So, imposing the constraint (1) is a
% sufficient condition for B(x) < 0 for all x in X0. (There is a slight
% technicality which we need to deal with and that I've ignored so far, which is
% that that we want strict inequality. This is the reason for the 1e-4 in the code
% below).

[prog,L1] = prog.newSOSPoly(monomials(x,0:d-2)); % Quadratic SOS multiplier polynomial
prog = prog.withSOS(-B - L1*g_X0 - 1e-4); % The 1e-4 is there to make sure the ...
                                           %inequality is strict (but you don't have to...
                                           %worry about this for the other constraints)


% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% C2: B(x) >= 0 inside unsafe set
[prog,L2] = prog.newSOSPoly(monomials(x,0:d-2)); % Quadratic SOS multiplier polynomial
prog = prog.withSOS(B - L2*g_Xu); % Similar reasoning as constraint C1

% C3: Bdot leq 0 everywhere
prog = prog.withSOS(-Bdot); % Since we want Bdot to be negative everywhere,
                            % we don't need a multiplier
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% DO NOT MODIFY THIS BLOCK OF CODE
% Solve SOS program
options = spot_sdp_default_options();
options.verbose = 1;
sol = prog.minimize(0,solver,options);

% Check is SOS program ran correctly
```

```
if ~sol.isPrimalFeasible
    error('The SOS problem is not feasible');
end

% One more check for SeDuMi
if strcmp(solver_name,'sedumi')
    if sol.info.solverInfo.feasratio < 0
        error('The SOS problem is not feasible');
    end
end


% Print out B after zeroing out very small coefficient
xprint = msspoly('x',2); % print in terms of x1 and x2
disp(' ');
disp('Barrier function:');
B_sol = clean(subs(sol.eval(B),x,xprint),1e-5)

% Plot stuff
figure
[X,Y] = meshgrid(linspace(-3,3,100),linspace(-3,3,100));
% initial condition set
gs_X0 = msubs(g_X0,x,[X(:),Y(:)]');
contour(X,Y,reshape(double(gs_X0),100,100),[0 0],'LineWidth',3); % initial condition set
hold on
% unsafe set
gs_Xu = msubs(g_Xu,x,[X(:),Y(:)]');
contour(X,Y,reshape(double(gs_Xu),100,100),[0 0],'r','LineWidth',3) % unsafe set
% 0-level set of B
gs_B = msubs(sol.eval(B),x,[X(:),Y(:)]');
contour(X,Y,reshape(double(gs_B),100,100),[0 0],'b','LineWidth',3) % unsafe set

% (scaled) vector field
[X,Y] = meshgrid(linspace(-3,3,50),linspace(-3,3,50));
x1dots = reshape(double(msubs(f(1),x,[X(:),Y(:)]')),50,50);
x2dots = reshape(double(msubs(f(2),x,[X(:),Y(:)]')),50,50);
x1dots = 0.1*x1dots./(sqrt(x1dots.^2 + x2dots.^2));
x2dots = 0.1*x2dots./(sqrt(x1dots.^2 + x2dots.^2));
quiver(X,Y,x1dots,x2dots,'AutoScale','off','Color','k');

% Title
title('Barrier functions')
xlabel('x_1');
ylabel('x_2');

legend('X_0','X_u','0 level-set of B(x)');
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Barrier Functions (Part D)

0.0/5.0 points (ungraded)

(d) Increase the radius of the initial condition set (without changing the point about which it is centered) to see how large you can make it and still guarantee that trajectories that start in it remain safe. When the initial condition set is too large, you will receive an error saying that "the SOS problem is not feasible". Type in the largest radius you found below (our tolerance on this answer will be 0.1). Note: remember to type in the radius and not the square of the radius!

Answer: 1.58

Submit You have used 0 of 3 attempts

Answers are displayed within the problem