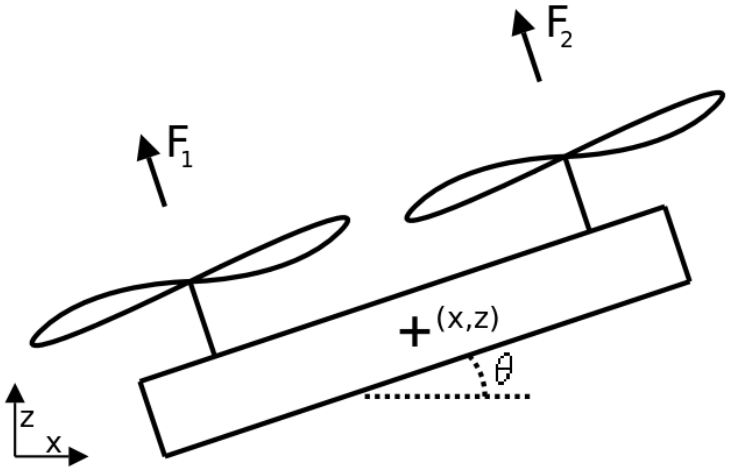
Problem: Definition of Underactuated

Definition of Underactuated

0.0/25.0 points (graded)

The following problem explores the definition of underactuated as described in the lecture notes. You should not need to derived detailed equations of motion for any of these problems.

Helicopter



A helicopter with two rotors is constrained to move in a vertical plane. Assume gravity acting on the helicopter. The task is to control the position (x, z) and pitch (θ) by varying the thrust produced by the two rotors. Decide whether this system is fully-actuated (in all states), or if there are any states in which the problem is underactuated. Use the definition of underactuated provided in lecture.

- Fully-actuated
- Underactuated ✔

Explanation

This is a case of trivial underactuation, where there are fewer actuators than degrees of freedom.

If you said "Underactuated" above, then please provide an expression for an acceleration that cannot be instantaneously achieved by the system. Assume that F_1 and F_2 are unbounded (and can be negative) and that the current state of the system is $x = 5, z = 1, \theta = 0.5$ radians. Your answer should consist of three **numerical values**, in the order $\ddot{x}\ \ddot{y}\ \ddot{\theta}$

xdd

Answer: cos(theta0)

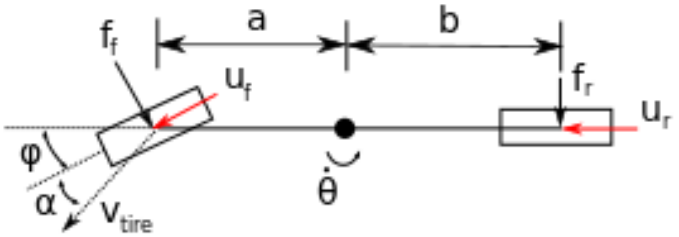
ydd

Answer: sin(theta0)

thetadd

Answer: [anything]

Bicycle Model



Consider the simple model of a vehicle known as the bicycle model, illustrated above. Let x, y be the position of the vehicle in inertial coordinates and θ be the heading angle. Lateral tire forces are typically modeled as being proportional to the lateral slip angle α, which defines the angle between the angle of the tire and the velocity of the tire v_{tire}, which depends on the speed and angular velocity of the vehicle, giving $f = C\alpha$.

Generating the equations of motion can be tedious, and we will often use a software package to do it automatically. For this problem, we have done it for you--but, as is often the case, the equations are pretty messy! One way to write the system of equations is:

$$
\begin{aligned}
f_r &= C_r \arctan\left(\frac{\dot{y}\cos\theta - \dot{x}\sin\theta - \dot{\theta}b}{\dot{x}\cos\theta + \dot{y}\sin\theta}\right) \\
f_f &= C_f\left(\arctan\left(\frac{\dot{y}\cos\theta - \dot{x}\sin\theta + \dot{\theta}a}{\dot{x}\cos\theta + \dot{y}\sin\theta}\right) - \phi\right) \\
I\ddot{\theta} &= -bf_r + a\left(f_f\cos\phi + u_f\sin\phi\right) \\
m\ddot{x} &= -f_r\sin\theta - f_f\sin\left(\theta+\phi\right) + u_r\cos\theta + u_f\cos\left(\theta+\phi\right) \\
m\ddot{y} &= f_r\cos\theta + f_f\cos\left(\theta+\phi\right) + u_r\sin\theta + u_f\sin\left(\theta+\phi\right)
\end{aligned}
$$

For the purposes of this problem, assume that the driver has control over the steering angle ϕ and has rear wheel drive. Treat the drive torque as a simple ground reaction force u_r acting at the tire and let $u_f = 0$. Is this system is fully-actuated or underactuated? Explain.

- Fully-actuated
- Underactuated ✔

Explanation
This is a case of trivial underactuation, where there are fewer actuators than degrees of freedom.

Now, suppose the the driver has control of both the front and rear longitudinal tire forces u_r and u_f and so has 3 total control inputs. In general, do you think this system is fully-actuated or underactuated? Give an intuitive explanation.

- Fully-actuated
- Underactuated ✔

Explanation
This vehicle model is still underactuated, despite that there are three actuators. Intuitively, the rotation of the vehicle and lateral translation are intrinsically coupled--for instance, the car cannot translate sideways without rotating.

Since these dynamics are not control affine, consider a simplified system of equations linearized about $\phi = \phi_0$ and $u_f, u_r = 0$. For simplicity, without loss of generality, let $\theta = 0$. Recalling the general dynamics form,

$\ddot{q} = f_1(q, \dot{q}, t) + f_2(q, \dot{q}, t)\, u$ We can write:

$$f_2(q, \dot{q}, t) = \begin{bmatrix} -aC_f\left(\left(\arctan\left(\frac{\dot{y}+\dot{\theta}a}{\dot{x}}\right) - \phi_0\right)\sin(\phi_0) + \cos(\phi_0)\right) & 0 & a\sin(\phi_0) \\ -C_f\left(\left(\arctan\left(\frac{\dot{y}+\dot{\theta}a}{\dot{x}}\right) - \phi_0\right)\cos(\phi_0) - \sin(\phi_0)\right) & 1 & \cos(\phi_0) \\ -C_f\left(\left(\arctan\left(\frac{\dot{y}+\dot{\theta}a}{\dot{x}}\right) - \phi_0\right)\sin(\phi_0) + \cos(\phi_0)\right) & 0 & \sin(\phi_0) \end{bmatrix}$$

Find the rank of f_2 when $\phi_0 = 0$.

Answer: 2

Is the system fully-actuated or underactuated?

- Fully-actuated
- Underactuated ✔

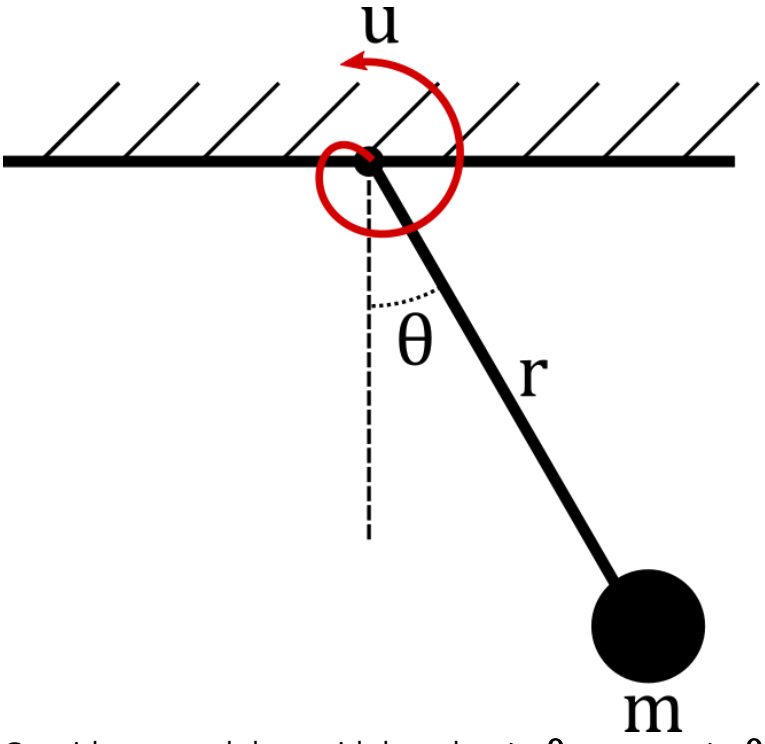
Are there values for ϕ_0 for which the system is fully-actuated? If so, write a symbolic expression of the form $g(\phi)$ where $g = 0$ describes these states. If there are no such states, enter 0. For grading purposes, ensure that $g(1) \geq 0$. For example, if the system is fully-actuated whenever $\phi_0 = 2$, write $g = 2 - \text{phi\_0}$.

$g =$

Answer: 0

Explanation
The first and third rows of the linearization are scalar multiples of one another, so f_2 will always be low rank.

Pendulum Model



Consider a pendulum with length $r > 0$, mass $m > 0$, and a single degree of freedom, θ. Let u be the torque applied to the pendulum, which will be our input to the system. Which of the following is/are true? (if none are true, then don't check any boxes).

- [ ] a. The system is underactuated.
- [x] b. The system is underactuated if u is bounded. ✔
- [ ] c. The system is underactuated if $\frac{\pi}{2} \leq \theta \leq \pi$
- [ ] d. The system is underactuated only when $\theta = \pm\pi$

Explanation

If u is unbounded, then for any acceleration $\ddot{\theta}_d$ we can find a corresponding torque u_d to produce that acceleration. That means that the pendulum is fully actuated when u is unbounded, regardless of the value of θ, so choices a, b, and d are all incorrect.

If u is bounded there will exist accelerations $\ddot{\theta}_d$ that cannot be produced without violating the bounds on u. That means that the pendulum is underactuated when u is bounded. Thus only choice b is correct.

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Problem: Feedback Linearization

Feedback Linearization

0.0/15.0 points (graded)

True or false: for any underactuated system of the form $\ddot{q} = f_1(q, \dot{q}) + f_2(q, \dot{q})\, u$, one can choose $u(x, u')$ so that $\dot{x} = Ax + Bu'$, where u' is a new control input.

- True
- False ✔

Take a robot whose dynamics are given by the manipulator equations, $H(q)\,\ddot{q} + C(q, \dot{q})\,\dot{q} + G(q) = B(q)\,u$ for $q \in \mathbb{R}^n$. The robot starts in a given initial configuration $q(0) = q_0$ and with a given initial velocity $\dot{q}(0) = \dot{q}_0$. Suppose $B(q)$ is rank n for all q. Which of the following statements are true for **any** twice-differentiable desired trajectory $q_d : \mathbb{R} \to \mathbb{R}^n$?

a. Feedback linearization can be used to make it so that $q(t) = q_d(t)$ for all $t \geq 0$

b. Feedback linearization can be used to make it so that $\dot{q}(t) = \dot{q}_d(t)$ for all $t \geq 0$

c. Feedback linearization can be used to make it so that $\ddot{q}(t) = \ddot{q}_d(t)$ for all $t \geq 0$

- [ ] a
- [ ] b
- [x] c ✔

Explanation
Feedback linearization of a second-order system can create arbitrary accelerations $\ddot{q}$, however, it cannot change the simple fact that we are dealing with a second-order system. Position and velocity are still limited in important, fundamental ways--particularly, both must be differentiable with respect to time. On the other hand, $\ddot{q}$ has no such restriction, it can, and often will be, discontinuous in time.

Oscillating Pendulum

Consider an actuated pendulum, where the base is forced to oscillate in simple harmonic motion, $C\sin(\omega t)$. Then, the dynamics of the pendulum angle θ are:

$$\ddot{\theta} = \frac{g}{l}\sin\theta - \frac{C}{l}\omega^2 \sin(\omega t)\sin\theta + \frac{u}{ml^2}$$

Even with the base shaking, we would like the pendulum to spin at a constant speed, $\dot{\theta} = 1$. To achieve this, we should choose $\ddot{\theta}_{des}$ to stabilize any velocity error. Use feedback linearization to find the control law such that $\ddot{\theta} = -\dot{\theta} + 1$. Write the variable v for $\dot{\theta}$.

Trigonometric functions and greek letters can be written out in English, and should format properly. For example, simply write "sin(omega*t)" to form $\sin(\omega t)$

$u =$

Answer: m*l^2*(-v+1-g/l*sin(theta) + C/l*omega^2*sin(omega*t)*sin(theta))

Explanation
Substituting the desired behavior, (\ddot \theta = -\dot \theta + 1\) into the dynamic equation above, we can rearrange and solve for u,

$$u = ml^2(1 - v) - mgl\sin(\theta) + mlC\omega^2 \sin(\omega t)\sin\theta$$

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Problem: Nonlinear Dynamics

Nonlinear Dynamics

0.0/20.0 points (graded)

Consider a system with the dynamics given by

$$\dot{x} = x^3 + 2x^2 - 5x - 6$$

Fill in the MATLAB code below, which is supposed to:

- Plot a phase diagram $\dot{x}$ vs. x for this system and set the three equilibrium points. Ensure that all equilibrium points are included in the plot range.
- Set the variable "eq_points" such that $x = eq\_points\,(i)$ is an equilibrium point.

Do not change the variable names x, xdot, and eq_points.

```
x =
xdot =
eq_points =
plot(x,xdot)

```

Unanswered

```
x = linspace(-10,10,1000);
xdot = x.^3 + 2*x.^2 - 5*x - 6;
eq_points = [-3;-1;2];
plot(x,xdot);
```

Run Code

Is the first equilibrium point (at the smallest value of x) stable, unstable, or marginally stable?

- Stable
- Unstable ✔
- Marginally stable

Is the second equilibrium point stable, unstable, or marginally stable?

- Stable ✔
- Unstable
- Marginally stable

Is the third equilibrium point (at the largest value of /(x/)) stable, unstable, or marginally stable?

- Stable
- Unstable ✔

Marginally stable

There is an interval, containing the origin, that is a region of attraction for one of these points. Identify this interval, using standard notation of (a, b) for open intervals and $[a, b]$ for closed intervals. Indicate a interval extending to infinity with "-inf" or "inf."

Answer: $\s * -\s * 3\s*, \s * 2\s*$

The region of attraction is (-3,2)

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Problem: Discrete Display

Discrete Systems

0.0/20.0 points (graded)

For a univariate dynamic system $\dot{x} = f(x)$ we have seen via graphical analysis that x^* is a locally stable equilibrium if the following conditions hold

1. $f(x^*) = 0$
2. $\frac{\partial f}{\partial x}(x^*) < 0$

Otherwise stated, that $f(x)$ has a zero-crossing at x^* with negative slope.

Now, consider a simple discretization of this continuous system, where for some fixed time step h we have:

$$x[k+1] = x[k] + hf(x[k])$$

For arbitrary h, the two conditions above are *not* sufficient for stability of the discrete system. Provide a counterexample demonstrating this by giving values for x_star, $f(x)$, and h below.

```
1 syms x
2 x_star =
3 h =
4 f =
5
```

Unanswered

```
% One possible solution is described
syms x
% pick a favorite system which is stable in the continuous sense
% any linear system will do. We'll also take the origin to be
% the equilibrium, for simplicity.
x_star = 0
f = -x

% now, let's try some values for h and see what happens
h = .1;
% simulate 100 steps, from x_0 close to 0
x_sim(1) = .01;
for i=2:100,
x_sim(i) = x_sim(i-1) + h*-x_sim(i-1);  % our discrete update rule
end
% x_sim(10) = 2.9e-7, so it looks like the discrete system is also stable
% try again with a much larger h*-x_sim
h = 10;
x_sim(1) = .01;
for i=2:100,
x_sim(i) = x_sim(i-1) + h*-x_sim(i-1);  % our discrete update rule
end
% Now x_sim(100) = -2.9e92, a massive number, so it's safe to say that this is unstable
```

Run Code

Find the upper bound h^* such that all $h < h^*$ results in a stable discrete system. Write your answer in terms of G, where $G = \left|\frac{\partial f}{\partial x}(x^*)\right| > 0$

Answer: 2/G

Explanation

Using graphical analysis for one dimensional continuous systems, we said that for an equilibrium to be stable, the dynamic flow had to point toward the equilibrium. Put another way, that the dynamics had to lead the system *closer* to the equilibrium. We saw this graphically by drawing arrows along the axis. Part 1 of this problem illustrated that if h is too large, the system can become unstable. What does it mean for the discrete system to get closer? It is, surprisingly, easier to formulate this criteria in the discrete case. Closer just means that $\left|x\left[k+1\right]-x^*\right|<\left|x\left[k\right]-x^*\right|$. For simplicity, we'll assume $x^*=0$. Substituting and simplifying, we get:

$$\begin{aligned}
\left|x\left[k+1\right]\right| &< \left|x\left[k\right]\right| \\
\left|x\left[k\right]+hf(x\left[k\right]\right| &< \left|x\left[k\right]\right| \\
\left(x\left[k\right]+hf(x\left[k\right])\right)^2 &< x[k]^2 \\
2hx\left[k\right]f\left(x\left[k\right]\right)+h^2f(x\left[k\right])^2 &< 0 \\
hf(x\left[k\right])^2 &< -2x\left[k\right]f\left(x\left[k\right]\right) \\
h &< -\frac{2x\left[k\right]}{f\left(x\left[k\right]\right)}
\end{aligned}$$

This must hold as $x\left[k\right]\to x^*$, so by linearization of f or through Taylor expansion, the right hand side simplifies to $\frac{2}{G}$. The Taylor expansion argument is given below:

$$\begin{aligned}
\lim_{x[k]\to 0} -\frac{2x\left[k\right]}{f\left(x\left[k\right]\right)} &= \lim_{x[k]\to 0} -\frac{2x\left[k\right]}{f\left(0\right)+x\left[k\right]f'\left(0\right)+.5x[k]^2f''\left(0\right)+\ldots} \\
&= \lim_{x[k]\to 0} -\frac{2}{f'\left(0\right)+.5x\left[k\right]f''\left(0\right)+\ldots} \\
&= -\frac{2}{f'\left(0\right)} \\
&= \frac{2}{G}
\end{aligned}$$

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Problem: Simple Pendulum

Simple Pendulum

0.0/10.0 points (graded)

The lecture introduced the simple pendulum as a benchmark nonlinear system. Recall that the second-order dynamics of a damped pendulum are

$$ml^2\ddot{\theta} + mgl\sin\theta = -b\dot{\theta} + u$$

Consider the case where the control input u takes on a constant value. Take the constants $m = 3$, $l = 1$, $g = 10$, and $b = 2$. Plot (but do not submit) the bifurcation diagram $\theta^* vs. u$ showing the equilibrium point(s) for a fixed u. Note what happens when u increases to 30 and above.

For $u = 10$, provide the equilibrium point(s) as a comma-separated list $\theta_1, \theta_2, \ldots$. Ensure that the number of equilibrium points is correct (no duplicate entries). The error tolerance for each element in the list is 10^{-3}. Restrict your answers to the interval $(-\pi, \pi]$.

Answer: 0.339837,2.801756

0.339837,2.801756

Do the same for $u = 30$.

Answer: 1.570796

1.570796

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Problem: Drake Installation

Drake Installation

0.0/10.0 points (graded)

Throughout the course, we will use the software toolbox Drake along with MATLAB. Drake contains our best implementations of many of the algorithms taught in the course, as well as code to generate the equations of motion from very simple descriptions of robots. For this Problem Set, we ask that you visit the website at http://drake.mit.edu and follow the installation procedure.

To verify that you were able to install Drake, within MATLAB, cd to the directory "examples/PlanarMonopodHopper" and enter the line "StateMachineControl.run". A simulation should run of a simple hopper. How many times did it hop?

Answer: 3

Please describe any problems you encountered during the installation process. To get the points for this problem, you have to write something!

1

Press ESC then TAB or click outside of the code editor to exit

Unanswered

Submit

Answers are displayed within the problem

Cost functions

Cost functions (Part a)

0.0/5.0 points (graded)

In this problem we will explore how to design cost functions that make the robot exihibit the kind of behavior we want. For this, we will consider the Dubins car model, which is a very simple model of a vehicle given by the following equations:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix}, \qquad \dot{\mathbf{x}} = f(\mathbf{x}, u) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} -\sin\psi \\ \cos\psi \\ u \end{bmatrix},$$

where $\mathbf{x}$ is the state of the system and consists of the states x (the x-position), y (the y-position) and ψ (the yaw angle of the vehicle). The control input is u.

(a) For optimal control problems, it is often useful to have cost functions of the form:

$$J = \mathbf{x}(t_f)^T Q_f \mathbf{x}(t_f) + \int_0^{t_f} g(t, \mathbf{x}(t))\, dt$$

where the first term depends only on where the robot ends up (the second term is simply the additive cost structure we saw in lecture). Here, t_f is the final time and Q_f is a symmetric positive semidefinite matrix. Suppose we want the robot to end up with its yaw angle close to 0, but do not care about the final x and y positions. What should we choose Q_f to be (remember to make sure it is symmetric and positive semidefinite)?

```
1 Qf =
2
```

Unanswered

```
% Since we don't care about the final x and y positions,
% Qf must have zeros everywhere except the (3,3) element.
% Since we want to penalize deviations of the yaw angles
% from 0 (and since our Qf must be p.s.d.), the (3,3) element
% must be positive. So, one possible answer is:

Qf = [0 0 0;0 0 0;0 0 1];
```

Run Code

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

Cost functions (Part b)

0.0/5.0 points (graded)

(b) Now suppose we want the vehicle to end up close to the line $y = 0.5x$, but we do not care exactly where on this line and what yaw angle it ends up in. What should we choose Q_f to be (remember to make sure it is symmetric and positive semidefinite)?

```
1 Qf =
2
```

Unanswered

```
% Since we don't care about the yaw angle, the third row and column
% must be all zeros. Hence, we only need to figure out the top left 2 by 2
% block. One possible cost function x'*Qf*x for penalizing deviations from
% the line y = 0.5 x is (y - 0.5x)^2. Thus, we can make x'*Q*x match this
% function by picking the top left 2 by 2 block correctly. We find that the
% following Qf achieves this:

Qf = [0.25 -0.5 0;-0.5 1 0;0 0 0];
```

Run Code

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

Cost functions (Part c)

0.0/5.0 points (graded)

(c) Now suppose we want to end up close to the curve $y = x^2$, and again do not care about the final yaw angle or where exactly on this curve we end up. Is it possible to achieve this given our setup?

- Yes
- No ✔

Explanation

One way to see it is to notice that $\mathbf{x^T Q_f x}$ can only be 0 along eigenvectors corresponding to 0 eigenvalues. Hence, it can only be 0 at the origin, or along straight lines, or along a plane. Thus, it cannot be 0 on the curve $y = x^2$.

Another way to see that this is not possible is to notice that $\mathbf{x^T Q_f x}$ is a (positive semidefinite) quadratic function of $\mathbf{x}$. Level-sets of positive semidefinite quadratic functions are either ellipsoids or straight lines (which are just "degenerate" ellipsoids). However, we want the cost to be 0 along the curve $y = x^2$ (which does not define an ellipsoid or a straight line).

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Linear Optimal Control

Linear Optimal Control

0.0/25.0 points (graded)

Consider the scalar equation

$$\dot{x} = -4x + 2u,$$

and the infinite horizon cost function

$$J = \int_0^\infty \left[32x^2 + u^2\right] dt.$$

(a) Assume that the optimal cost-to-go function is of the form $J^\star = px^2$. What value of p satisfies the Hamilton-Jacobi-Bellman conditions for optimality?

Answer: 2.0

Explanation

Given the equation of motion

$$\dot{x} = -4x + 2u,$$

the infinite horizon cost function, $J = \int_0^\infty \left[32x^2 + u^2\right] dt$ and the form of the cost-to-go ($J^* \left(x\right) = px^2$), we can substitute into the Hamilton-Jacobi-Bellman equation as follows. We know:

$$0 = \min_u \left\{ g\left(\mathbf{x}, \mathbf{u}\right) + \frac{\partial J^*}{\partial x} \mathbf{f}\left(\mathbf{x}, \mathbf{u}\right) \right\}$$

We also know that

$$g(\mathbf{x},\mathbf{u}) = 32x^2 + u^2$$
$$\mathbf{f}(\mathbf{x},\mathbf{u}) = -4x + 2u$$
$$\frac{\partial J^*}{\partial x} = 2px$$
$$\frac{\partial J^*}{\partial t} = 0$$

Substituting in we get:

$$0 = \min_u \left\{32x^2 + u^2 + (2px)(-4x + 2u)\right\} \quad (Eq.\,1)$$

Now, let

$$\mathcal{H} = 32x^2 + u^2 + (2px)(-4x + 2u)$$

We know that for u to minimize $\mathcal{H}$, we must have $\frac{\partial \mathcal{H}}{\partial u} = 0$.

$$\frac{\partial \mathcal{H}}{\partial u} = 2u + (2px)(2)$$

We can now see that the optimal u, $u^\star$, is $-2px$. Plugging back in to Eq. 1, we get

$$0 = \left\{32x^2 + (-2px)^2 + (2px)(-4x + 2(-2px))\right\}$$

Which simplifies to:

$$0 = 32x^2 - 4p^2x^2 - 8px^2$$

Because this must be true for all x, this reduces to following quadratic form:

$$0 = -4p^2 - 8p + 32$$

From this we know that $p = \{2, -4\}$. Because J^* should be positive definite we choose $p = 2$.

(b) Given that the optimal feedback controller associated with $J^\star$ is $u^\star = -Kx$, what is the value of K?

Answer: 4.0

Explanation
Once we have solved for p as in part (a), we can easily find $u^\star$. From before we know that $u^\star = -2px$, so $u^\star = -Kx$ where $K = 4$.

(c) Suppose we change our cost to the following:

$$J = \int_0^\infty \left[96x^2 + 3u^2\right] dt.$$

Which of the following statements is true? (Select all that apply)

- [ ] The optimal controller (K) gets multiplied by 3
- [ ] The optimal controller (K) gets divided by 3
- [x] The optimal cost-to-go gets multiplied by 3
- [ ] The optimal cost-to-go gets divided by 3
- [ ] None of the above

Explanation

The optimal cost-to-go gets multiplied by 3, but the controller does not change. This can be easily verified by going through the same calculations as in part (a) and (b). This is a feature of LQR (Linear Quadratic Regulator) problems in general. Scaling both the cost on state and action does not change the controller (but only changes the optimal cost-to-go).

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

Value Iteration (Double Integrator)

Value Iteration (Double Integrator)

0.0/30.0 points (graded)

In this problem, we'll consider the optimal control problem for the double integrator (unit mass brick on ice), described by

$$\ddot{q} = u, \ |u| \leq 1$$

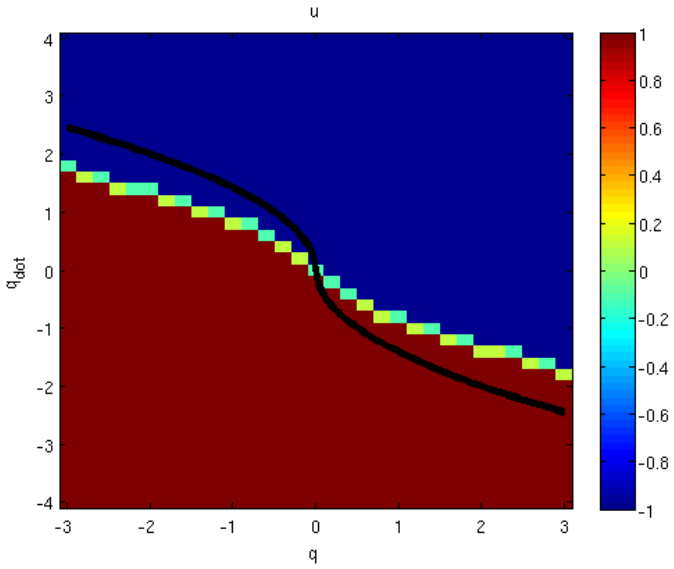
using the Value Iteration algorithm. An implementation of that algorithm is available for you in Drake (see the runValueIteration function in examples/DoubleIntegrator.m). This is a complete implementation of the algorithm with discrete actions and volumetric interpolation over state.

(a) Run the value iteration code for the double integrator to compute the optimal policy and optimal cost-to-go for the minimum-time problem. Compare the result to the analytical solution we found in lecture (also available in Example 9.2 in the course notes) by answering the following questions.

1) Find an initial condition of the form $(2, \dot{q}_0)$ such that the value iteration policy takes an action in exactly the wrong direction from the true optimal policy. Type in your value of $\dot{q}_0$ below:

Answer: -1.7

Explanation



The controller obtained from value iteration is plotted above. The black curve is the switching surface for the analytically derived optimal controller (i.e., where the control input switches from -1 to +1). The value iteration controller has the wrong sign at all points lying in the "band" between the switching surface for the analytical controller and the switching surface for the value iteration controller. One such point is $(2, -1.7)$.

2) What is the true optimal time-to-go from this state (i.e., for the optimal bang-bang controller derived in class)?

Answer: 2.0

Explanation

We derived the optimal cost-to-go function in class. For our point $(2, -1.7)$, the optimal cost is given by $2\sqrt{(\dot{q}_0^2)/2 + q_0} + \dot{q}_0$, which is approximately 2.01.

3) What is the time-to-go from this state estimated by value iteration?

Answer: 4.1390

Explanation

This can be read off from the plot generated by the value iteration code (the "index" field in the cost-go-go plot). It is approximately 4.14.

4) When implementing value iteration, one needs to be wary of several implementation details. Find a setting of the discretization (i.e., the variable xbins in DoubleIntegrator.m) that causes the code to NOT converge. The maximum distance between points in the q and $\dot{q}$

directions should still be at most 0.2, and the grid must still contain the square with sides of length 2 centered about the origin. (Hint: it might help to see how the minimum-time cost function is implemented).

```
q_bins =
qdot_bins =
xbins = {q_bins,qdot_bins};

```

Unanswered

```
% Any solution where the grid does not contain the origin will cause
% the value iteration to not converge. This is because the cost-to-go
% is never 0 at any point. So, one possible answer is:
q_bins = linspace(-3,3,40);
qdot_bins = linspace(-3,3,40);
xbins = {q_bins,qdot_bins};
```

Run Code

(b) Change the cost-to-go function to a combination of the quadratic regulator problem and the minimum-time problem:

$$g(q, \dot{q}, u) = c(q, \dot{q}) + Q_p q^2 + Q_d \dot{q}^2 + R u^2,$$

where $c(q, \dot{q})$ is 0 when $(q, \dot{q}) = (0, 0)$ and 1 otherwise. Use $Q_p = Q_d = 1, R = 10$.

1) What is the cost-to-go from the point $(1.0, 1.0)$ estimated by the value iteration?

Answer: 31.14

Explanation
This can be read off from the plot generated by the value iteration code (the "index" field in the cost-go-go plot). It is approximately 31.14.

2) When implementing controllers on real robots, we have to be mindful of the fact that our models seldom capture all aspects of the behavior of the system. Supposing that in addition to the constraints imposed on the maximum and minimum control input (i.e., $|u| \leq 1$), our real physical brick "robot" also had constraints on the derivative of the control input (let's say $|\dot{u}| \leq 1$). Assuming that you only cared about stabilizing the system to the origin, which controller would you prefer to implement?

- Minimum-time (cost from part (a))
- Quadratic cost + minimum-time (cost from part (b)) ✔

Explanation
As discussed in class, the minimum-time cost typically yields controllers that change very sharply from -1 to 1 along the switching surface. This would violate the constraint $|\dot{u}| \leq 1$. The quadratic cost adds a penalty to the actions and causes them to get smoothed out (as you can see from the controller found by value iteration in part (b)).

Submit You have used 0 of 3 attempts

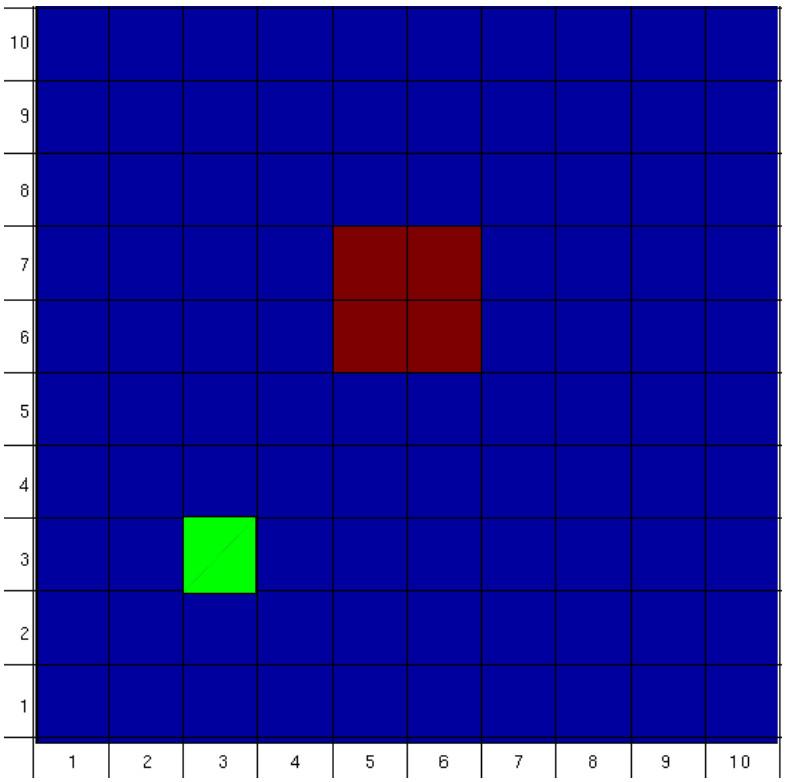
Answers are displayed within the problem

Value Iteration (Stochastic Grid World)

Value Iteration (Stochastic Grid World)

0.0/30.0 points (graded)

In this problem we will consider the optimal control problem for a stochastic version of the "Grid World" domain we examined in lecture. Again, all the Value Iteration code is provided in Drake. Your task will be to setup the transition matrices and cost functions in order to define the stochastic dynamics and optimal control problem.



The stochastic Grid World consists of a $n \times n$ grid (see figure above). We will take $n = 10$ for this example. Here, the cells are listed in (x, y) order. The cell (1,1) corresponds to the bottom left corner, the cell (10,1) corresponds to the bottom right, and the cell (10,10) corresponds to the top right. The actions that the robot can take are "up", "right", "down", and "left". The dynamics in this domain are as follows:

- If the robot is on any cell at the border of the domain and attempts to take a move that would take it out of the domain, it will remain in the same cell with probability 1. If it is on the border and attempts to move in any other direction (i.e., one that will not take it out of the domain), it will move in that direction with probability 1. So, for example, if the robot is in cell (4,10), and it takes the "up" action, it will remain in cell (4,10) with probability 1. If it takes the "right" action, it will move to cell (5,10) with probability 1.

- If the robot is on a cell that is not on the border and attempts to move in a given direction, it will successfully do so with probability p. With probability $(1-p)/3$ it will move in any of the other three directions.

- If the robot is at the goal state, the robot will remain there with probability 1 no matter what action it takes.

The goal is to get to the cell (3,3), while avoiding an obstacle that spans the four cells (5,6), (5,7),(6,6),(6,7) as shown in the picture above. In order to achieve this, we will make our cost function as follows:

- The cost of being in any of the obstacle cells and taking any action from these is 50.

- The cost of being at the goal state and taking any action is 0.

- In any other state, the cost of taking any action is 1.

(a) Take a look at the "properties" in the MarkovDecisionProcess.m file in the drake/systems/ folder. Write Matlab code to compute the transitions T and cost C that are required as inputs to this function, corresponding to the transition dynamics and cost described for the stochastic grid world above. (The states S and actions A are already provided for you below). We will take the probability p described above to be 0.75.

Some helpful tips: (1) You should test out your code on a local machine before copying it below. (2) It might be helpful to test your code out with $p = 1$ first since the solution there is easy to verify (however, be sure to change p back to 0.75 when making the submission!). (3) It might be useful to obtain solutions to the entire problem (i.e., parts (b) and (c) before submitting your code below).

```
% States and actions
n = 10;
N = n^2;
[y,x] = meshgrid(1:n,1:n);
S = [x(:)';y(:)'];
A = [1,2,3,4]; % 1:"up", 2:"right", 3:"down", 4:"left"

% Costs
```

```
C = sparse(N,4);

% Transitions
T{1} = sparse(N,N);
T{2} = sparse(N,N);
T{3} = sparse(N,N);
T{4} = sparse(N,N);
```

Unanswered

```
% States and actions
n = 10;
N = n^2;
% S = 1:N;
[y,x] = meshgrid(1:n,1:n);
S = [x(:)';y(:)'];
A = [1,2,3,4];

% Costs
C = sparse(N,4);
% Obstacles
for i = 1:n
    for j = 1:n
        % Obstacles (cost should be 50)
        if all([i,j] == [5,6]) || all([i,j] == [6,6]) || all([i,j] == [5,7]) || all([i,j] == [6,
            C((j-1)*n+i,:) = 50*ones(1,4);
        else
            % Cell is not an obstacle (cost should be 1)
            C((j-1)*n+i,:) = ones(1,4);
        end
    end
end

% Goal (3,3), cost should be 0
C(2*n+3,:) = zeros(1,4);


% Transition matrices
T{1} = sparse(N,N);
T{2} = sparse(N,N);
T{3} = sparse(N,N);
T{4} = sparse(N,N);

% Probabilities of transitions
p_correct = 0.75;
p_wrong = (1-p_correct)/3;

% Action = 1 (up)
for i = 1:N
    if any(i == [(N-n+1):N])
        % on the top border
        T{1}(i,i) = 1;
    elseif (mod(i,n) == 0)
        % right border
        T{1}(i,i+n) = 1;
    elseif any(i == [1:n])
        % bottom border
```

```
        T{1}(i,i+n) = 1;
    elseif (mod(i,n) == 1)
        % left border
        T{1}(i,i+n) = 1;
    else
    % Typical case (not on the border)
    T{1}(i,i-n) = p_wrong;
    T{1}(i,i+1) = p_wrong;
    T{1}(i,i+n) = p_correct;
    T{1}(i,i-1) = p_wrong;
    end
end

% Action = 2 (right)
for i = 1:N
    if (mod(i,n) == 0)
        % on the right border
        T{2}(i,i) = 1;
    elseif any(i == [(N-n+1):N])
        % top border
        T{2}(i,i+1) = 1;
    elseif any(i == [1:n])
        % bottom border
        T{2}(i,i+1) = 1;
    elseif (mod(i,n) == 1)
        % left border
        T{2}(i,i+1) = 1;
    else
    % Typical case (not on the border)
    T{2}(i,i-n) = p_wrong;
    T{2}(i,i+1) = p_correct;
    T{2}(i,i+n) = p_wrong;
    T{2}(i,i-1) = p_wrong;
    end
end

% Action = 3 (down)
for i = 1:N
    if any(i == [1:n])
        % on the bottom border
        T{3}(i,i) = 1;
    elseif (mod(i,n) == 0)
        % right border
        T{3}(i,i-n) = 1;
    elseif any(i == [(N-n+1):N])
        % top border
        T{3}(i,i-n) = 1;
    elseif (mod(i,n) == 1)
```

```
        % left border
        T{3}(i,i-n) = 1;
    else
    % Typical case (not on the border)
    T{3}(i,i-n) = p_correct;
    T{3}(i,i+1) = p_wrong;
    T{3}(i,i+n) = p_wrong;
    T{3}(i,i-1) = p_wrong;
    end
end

% Action = 4 (left)
for i = 1:N
    if (mod(i,n) == 1)
        % on the left border
        T{4}(i,i) = 1;
    elseif (mod(i,n) == 0)
        % right border
        T{4}(i,i-1) = 1;
    elseif any(i == [1:n])
        % bottom border
        T{4}(i,i-1) = 1;
    elseif any(i == [(N-n+1):N])
        % top border
        T{4}(i,i-1) = 1;
    else
    % Typical case (not on the border)
    T{4}(i,i-n) = p_wrong;
    T{4}(i,i+1) = p_wrong;
    T{4}(i,i+n) = p_wrong;
    T{4}(i,i-1) = p_correct;
    end
end

% Once you're at the goal, you stay there
for i = 1:N
    T{1}(2*n+3,i) = 0;
    T{2}(2*n+3,i) = 0;
    T{3}(2*n+3,i) = 0;
    T{4}(2*n+3,i) = 0;
end

T{1}(2*n+3,2*n+3) = 1;
T{2}(2*n+3,2*n+3) = 1;
T{3}(2*n+3,2*n+3) = 1;
T{4}(2*n+3,2*n+3) = 1;
```

Run Code

(b) Download the file StochasticGridWorld.m from here. This file calls the Value Iteration code in Drake and sets up the visualization for the stochastic grid world. You will not need to modify this file. All you need to do is create a function stochasticSATC() that outputs S,A,T, and C as computed in part (a) above:

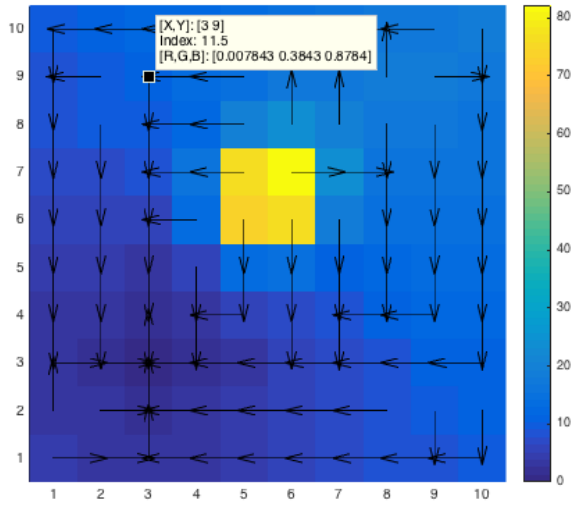
[S,A,T,C] = stochasticSATC()

Make sure to save the stochasticSATC.m file in the same folder as StochasticGridWorld.m. Now run the runValueIteration function in StochasticGridWorld.m. This will run value iteration using the transition dynamics and cost function you defined.

What is the optimal cost-to-go returned by value iteration at the cell (3,9)?

Answer: 11.5

Explanation

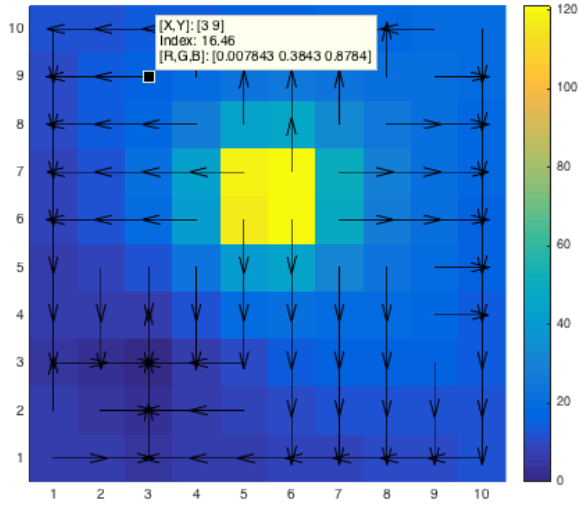


This can be read off from the plot generated by the code (the "index" field). The cost-to-go is approximately 11.5.

(c) Change the probability p to 0.5. Now what is the optimal cost-to-go returned by value iteration at the cell (3,9)?

Answer: 16.46

Explanation



This can be read off from the plot generated by the code (the "index" field). The cost-to-go is approximately 16.46.

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Controllability and Stability

Stability and Controllability

0.0/18.0 points (graded)

For this problem, answer a series of questions related to some of the basic notions discussed in lecture.

Cubic Plant

Consider the one-dimensional system $\dot{x} = -x^3$. Using graphical analysis, is the trivial equilibrium stable or unstable?

- stable ✔
- unstable
- cannot tell

What is the eigenvalue of the linearized system?

Answer: 0

Does the linearization indicate stability?

- stable
- unstable

- cannot tell ✔

Explanation
Graphical analysis shows that $\dot{x} < 0$ for $x > 0$ and $\dot{x} > 0$ for $x < 0$, so the equilibrium at the origin is stable. However, the linearization is uninformative, since the eigenvalue of the linearization is zero.

Controllability and Underactuation

Select any true statement.

- [ ] A system that is controllable is fully-actuated.
- [ ] A system that is fully-actuated is controllable. ✔

Explanation
Fully-acutated sytems are trivial examples of controllable systems. However, many underactuated systems (some of which we will see in the course) are also controllable!

Linear Quadratic Regulators

Suppose we have a controllable system, where the linearization is $\dot{x} = Ax + Bu$. For $Q \succ 0$ and $R \succ 0$, we can find the LQR controller with feedback gain matrix K.

What can we say about the eigenvalues of A?

- All eigenvalues have negative real part
- Some eigenvalues have a positive real part
- It depends on the system ✔

What can we say about the eigenvalues of $A - BK$?

- All eigenvalues have negative real part ✔

- Some eigenvalues have a positive real part
- It depends on the system

Explanation

Controllability, by itself, does not say anything about the stability of the system (or the eigenvalues of A). However, LQR is guaranteed to stabilize any controllable system. Therefore, the closed-loop dynamics $\dot{x} = (A - BK)x$ are stable.

Submit You have used 0 of 1 attempt

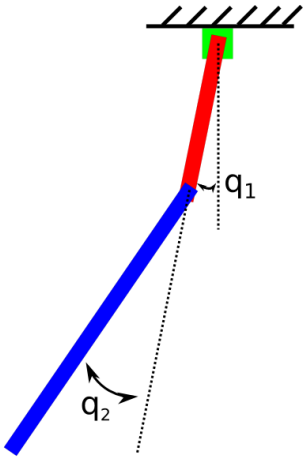
Answers are displayed within the problem

Acrobot Balancing

Acrobot Linearization

0.0/10.0 points (graded)

In this problem, we will develop an end-to-end controller capable of swinging up the acrobot and balancing it at the upright configuration.



First, let's use LQR to design a stabilizing controller for the upright configuration $q_0 = \begin{bmatrix} \pi \\ 0 \end{bmatrix}$.

The first step is to linearize the system about the upright point. Find the matrices A and B such that the linearization is

$$\frac{d}{dt}\begin{bmatrix} q \\ \dot{q} \end{bmatrix} \approx A \begin{bmatrix} (q - q_0) \\ \dot{q} \end{bmatrix} + Bu$$

With the linearization, use MATLAB's **[K,S] = lqr(A,B,Q,R)** function for some symmetric, positive definite cost matrices Q and R of your choosing.

It is recommended that you find the answers using your local copy of MATLAB and paste in the results below.

Drake Acrobot

In the **examples/Acrobot** directory of Drake, you will also find the Acrobot model **Acrobot.urdf**. Note that **AcrobotPlant** class uses different mass and length parameters! Don't use it!

You will find the following lines of code useful:

```
p = PlanarRigidBodyManipulator('Acrobot.urdf');
[f,df] = p.dynamics(t,x,u);
```

Which calculates $\dot{x} = f(t, x, u)$ and its derivative df given by

$$df = \begin{bmatrix} \frac{\partial f}{\partial t} & \frac{\partial f}{\partial x} & \frac{\partial f}{\partial u} \end{bmatrix}$$

Since the acrobot is time-invariant, we expect $\frac{\partial f}{\partial t} = 0$.

Note that the gradients (as are all gradients of vector-valued functions in Drake) are given in the following form,

$$df(:, i) = \frac{\partial f}{\partial x_i}$$

Alternatively, you might inspect the function **linearize** within the **DrakeSystem** class for an even easier approach.

Submission

Enter your answer with A, B, Q, R, K and S below. Please **DO NOT** attempt to use Drake functions in your answer. Calculate A, B in your local copy of MATLAB and then paste the answer below. You may find the mat2str() function helpful for converting a matrix into something suitable to be copied and pasted into the answer box.**lqr()** is a base MATLAB function, and so you can use it in your solution.

```
A =
B =
Q =
R =
[K,S] = lqr(A,B,Q,R);

```

Unanswered

```
% One option to find A and B using Drake is:
% p = PlanarRigidBodyManipulator('Acrobot.urdf');
% [f,df] = p.dynamics(t,x,u);
% A = f(:,2:5);
% B = f(:,6);

A = [0 0 1 0;0 0 0 1;12.6292 -12.6926 -.1721 .3015; -14.75 29.61 .3015 -.6033];
B = [0;0;-3.01;6.033];
Q = eye(4);
R = 1;
[K,S] = lqr(A,B,Q,R);
```

Run Code

LQR is just that easy! Now you have a controller that will stabilize the equilibrium at q_0. In terms of x, x_0, A, B and K *only* write the expression for LQR controller u.

u=

Answer: -K*(x-x_0)

Explanation
Since we linearized about the nominal point x_0, the LQR control law is $-K\left(x - x_0\right)$.

Download the stub code for an acrobot controller here and save it into the **examples/Acrobot** folder within your installation of Drake. When saving the file, make sure to NOT save it as a complete web page, or you will end up with some unwanted HTML headers. Complete the **output** function with your LQR controller, and run the simulation from a few initial states near the upright (see **AcrobotController.run**). Try a few different values for Q and R and note how the performance changes.

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Acrobot Partial Feedback Linearization

Acrobot Partial Feedback Linearization

0.0/10.0 points (graded)

If, on the previous problem, you did not try simulating your LQR controller from initial states far away from the upright, try it now. It doesn't work so well, does it?

As the next step to developing a swing-up controller, we will use partial feedback linearization to regulate the elbow joint, q_2. Recall the manipulator equations:

$$\ddot{q} = H^{-1}\left(Bu - C\right)$$

Note that C is a vector here. If this form of the manipulator equation looks unfamiliar, you can just pre-multiply both sides by H and rearrange the terms to get back to the form we saw in lecture.

For some desired value y, use partial feedback linearization to determine u such that $\ddot{q}_2 = y$.

For the acrobot, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Writing $C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$ and $H^{-1} = \begin{bmatrix} a_1 & a_2 \\ a_2 & a_3 \end{bmatrix}$, find u in terms of y, C_1, C_2, a_1, a_2 and a_3.

u=

Answer: y/a_3 + C_2 + a_2/a_3*C_1

Explanation

Substitute in $\ddot{q}_2 = y$ and examine the second row of the dynamics:

$$\ddot{q}_2 = y = a_3 u - a_2 C_1 - a_3 C_2$$

$$u = \frac{1}{a_3}(a_2 C_1 + a_3 C_2 + y)$$

$$u = \frac{1}{a_3}(a_2 C_1 + y) + C_2$$

Did you use collocated or non-collocated partial feedback linearization for this problem?

- collocated ✔
- non-collocated

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Acrobot Energy Shaping

Acrobot Energy Shaping

0.0/5.0 points (graded)

The last piece of the puzzle is to implement an energy shaping algorithm for swing-up.

Setting aside (for now), your partial feedback linearization, use a similar energy shaping controller to the one which worked for the pendulum. For some user-defined gain k_1 and the total potential energy at the upright position E_d, use the control input $u = k_1 \left(E_d - E\right) \dot{q}_2$ so that $\dot{E} = k_1 \left(E_d - E\right) \dot{q}_2^2$.

Implement this controller in the stub code **AcrobotController.m**, and try a few different values for k_1. Simulate the system from states near the downright (passively stable) equilibrium. What is the typical behavior? Answer all that apply.

- [ ] Energy approaches the desired energy ✔
- [ ] Energy does not approach the desired energy
- [ ] The state approaches the upright equilibrium
- [ ] The state does not approach the upright equilibrium ✔

Explanation

The energy controller should succeed in regulating total energy, but this is not enough to get to the upright equilibrium. For the acrobot, the set of states with constant energy is three-dimensional, so the energy shaping controller does not reach a homoclinic orbit.

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Acrobot Swingup

0.0/30.0 points (graded)
At last, we're going to put it all together.

First, combine the energy shaping controller with your partial feedback linearization. Choose $y = -k_2 q_2 - k_3 \dot{q}_2$ to stabilize the q_2 coordinate. If u_p is the resulting partial feedback linearization controller, and u_e is the energy shaping controller above, implement the control law $u = u_p + u_e$.

Simulate this control law for a few trajectories, starting from near the downward position, trying a few values for k_1, k_2 and k_3. Unlike with the pendulum, we do not have a proof that we will reach an orbit that passes near to the upright position. However, if you've chosen reasonable gains, you should find that the acrobot does, in fact, swing up--but that it then quickly falls back down. We're still missing stabilization!

This is where the LQR controller from earlier comes into play. The tough question here is when to switch from the swingup controller to the stabilization controller? Ideally, we would like to switch whenever the current state is in the region of attraction created by the LQR solution. However, determining this region is quite hard in practice, and something we will talk about later on in the course.

For now, the LQR cost-to-go function, $(x - x_0)^T S (x - x_0)$ gives a good notion of how "far" away any state is from the equilibrium x_0. In your MATLAB controller, implement a case statement to switch from the swing-up controller to the LQR controller, whenever this cost-to-go is within some threshold.

Now, you are good to go! There are a few values to tune: Q, R, k_1, k_2, k_3 and the switching threshold. You may have to play around with these values to get a stabilizing controller, but once you have something reasonable, the solution should be quite robust.

Once you are satisfied with your controller, record the output from one run by executing

```
[t,x,x_grade]=AcrobotController.run;
mat2str(x_grade)
```

Copy and paste the state trajectory **x_grade** into the answer window below. Your solution should result in **x_grade** being a 100x4 matrix. The matlab function

```
mat2str
```

just converts the value of x_grade into a string which, when pasted into matlab, results in x_grade being correctly constructed. It is recommended that you put your answer into a local .m file on your computer and ensure that it will run locally.

```
x_grade = [
% paste here
];

```

Unanswered

```
% controller code
[f,df] = obj.p.dynamics(0,[pi;0;0;0],0);
A = df(:,2:5);
B = df(:,end);
[K,S] = lqr(A,B,eye(4),1);
xe = [q;qd] - [pi;0;0;0];

if  xe'*S*xe < 1000
  u = K*([pi;0;0;0] - [q;qd]);
else
  % some useful functions + properties

  % acceleration due to gravity
  g = 9.81;
  m = obj.p.getMass;

  % computes center of mass [x;z] at any position
  com = obj.p.getCOM(q);

  % kinetic energy of a manipulator is 1/2 * qdot^T * H * qdot
  [H,C,B] = obj.p.manipulatorDynamics(q,qd);

  E_des = m*g*1.25;
  E_current = m*g*com(2) + .5*qd'*H*qd;

  M22bar = H(2,2) - H(2,1)/H(1,1)*H(1,2);
  h2bar = C(2) - H(2,1)/H(1,1)*C(1);
  k1 = 10;
  k2 = 10;
  k3 = 1;
  ubar = (E_des - E_current)*qd(2);

  u = M22bar*(-k1*q(2) - k2*qd(2)) + h2bar  + k3*ubar;
end
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

OPTIONAL: LQR with discounting

LQR with discounting

0.0/10.0 points (ungraded)

In this problem, we will consider a variation on the continuous Linear Quadratic Regular introduced in class.

Consider a discrete-time linear system: $\boldsymbol{x}\left[\boldsymbol{k}+1\right]=\boldsymbol{A}\boldsymbol{x}\left[\boldsymbol{k}\right]+\boldsymbol{B}\boldsymbol{u}\left[\boldsymbol{k}\right]$. Suppose we are more interested in the near-term behavior of the system, so we introduce a discount factor into the cost. The cost over a finite horizon of $\boldsymbol{N}$ total steps can be written as

$$\sum_{k=0}^{N}\alpha^{k}\left(x^{T}Qx+u^{T}Ru\right)$$

Note that, because of the discount factor, we can expect the cost to be dependent on both $\boldsymbol{x}$ and $\boldsymbol{k}$. You are given that the cost-to-go at time $\boldsymbol{k}$ will be given by $\boldsymbol{J_k}\left(\boldsymbol{x}\left[\boldsymbol{k}\right]\right)=\boldsymbol{x}[\boldsymbol{k}]^{\boldsymbol{T}}\boldsymbol{S_k}\boldsymbol{x}\left[\boldsymbol{k}\right]$. We will use the discrete-time version of the HJB equation to derive the Riccati equation describing the optimal cost-to-go matrix $\boldsymbol{S_0}$.

The applicable version of the HJB equation to this problem is given below:

$$\boldsymbol{J_k}\left(\boldsymbol{x}\left[\boldsymbol{k}\right]\right)=\min_{\boldsymbol{u}[\boldsymbol{k}]}\boldsymbol{g_k}\left(\boldsymbol{x}\left[\boldsymbol{k}\right],\boldsymbol{u}\left[\boldsymbol{k}\right]\right)+\boldsymbol{J_{k+1}}\left(\boldsymbol{x}\left[\boldsymbol{k}+1\right]\right)$$

Substitute the given terms for $\boldsymbol{g_k}$, $\boldsymbol{J_{k+1}}$ and $\boldsymbol{x}\left[\boldsymbol{k}+1\right]$ and solve for the optimal controller in terms of $\boldsymbol{\alpha}$, $\boldsymbol{k}$, $\boldsymbol{R}$, $\boldsymbol{B}$, $\boldsymbol{S_{k+1}}$, $\boldsymbol{A}$ and $\boldsymbol{x}$. Please leave the template code in the solution, and fill in the expression for $\boldsymbol{u}$.

```
alpha = .85;
k = randi(100);
R = diag(rand(2,1));
B = randn(4,2);
A = randn(4);
S_k1 = 100*diag(rand(4,1));
x = randn(4,1);
```

```
% your solution to u below
u = ;

```

Unanswered

```
alpha = .85;
k = randi(100);
R = diag(rand(2,1));
B = randn(4,2);
A = randn(4);
S_k1 = 100*diag(rand(4,1));
x = randn(4,1);

u = -inv(alpha^k * R + B'*S_k1*B)*B'*S_k1*A*x;
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

LQR with discounting, part 2

0.0/10.0 points (ungraded)

Substitute your solution to the previous part into the HJB equation and find the algebraic Riccati equation relating S_{k+1} and S_k

Substitute the given terms for g_k, J_{k+1} and $x\left[k+1\right]$ and solve for the optimal controller in terms of α, k, R, B, S_{k+1} and A.

While the derivation will look messy at first, it should simplify to the form $S_k = \alpha^k Q - M + A^T S_{k+1} A$ for some invertible matrix M (note the signs in the previous expression).

As with the previous part, please leave the template code in the solution, and fill in the expression for M.

```
alpha = .85;
k = randi(100);
R = diag(rand(2,1));
B = randn(4,2);
A = randn(4);
S_k1 = 100*diag(rand(4,1));
% your solution to u below
M = ;

```

Unanswered

```
alpha = .85;
k = randi(100);
R = diag(rand(2,1));
B = randn(4,2);
A = randn(4);
S_k1 = 100*diag(rand(4,1));

M = A'*S_k1*B*inv(alpha^k*R + B'*S_k1*B)*B'*S_k1*A;
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

LQR with discounting, part 3

0.0/7.0 points (ungraded)

This finite-horizon discrete LQR can be easily solved backwards in time, given a choice for the final cost S_N.

Suppose we have the system:

$$A = \begin{bmatrix} 2 & 1.5 \\ -3 & 2 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

And choose costs:

$$Q = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}, \qquad R = 1, \qquad S_N = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

With $\alpha = .85$ and $N = 10$, find S_0;

```
alpha = .85;
N = 10;
A = [2 1.5; -3 2];
B = [0;1];
Q = diag([5;1]);
R = 1;
S_N = 10*eye(2);
S_0 =

```

Unanswered

```
alpha = .85;
N = 10;
A = [2 1.5; -3 2];
B = [0;1];
Q = diag([5;1]);
R = 1;
S_N = 10*eye(2);

S{N+1} = S_N;
for k=N-1:-1:0,
  S_k1 = S{k+2};
  M = A'*S_k1*B*inv(alpha^k*R+B'*S_k1*B)*B'*S_k1*A;
   S{k+1} = alpha^k*Q - M + A'*S_k1*A;
end
S_0 = S{1}
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Lyapunov Functions

Lyapunov Functions

0.0/20.0 points (graded)

This question will test your understanding of Lyapunov functions with a series of short questions.

(a) Suppose that $V_1(x)$ and $V_2(x)$ are valid Lyapunov functions that prove global stability of a system to the origin. Is it always the case that $V_2(x) = cV_1(x)$ for some $c > 0$? In other words, are Lyapunov functions unique up to scaling?

- Yes
- No ✔

Explanation

Lyapunov functions are not unique up to scaling. As an example, consider the system $\dot{x} = -x$. The function $V(x) = x^2$ is a valid Lyapunov function, but so is $V(x) = x^4$.

(b) For the system:

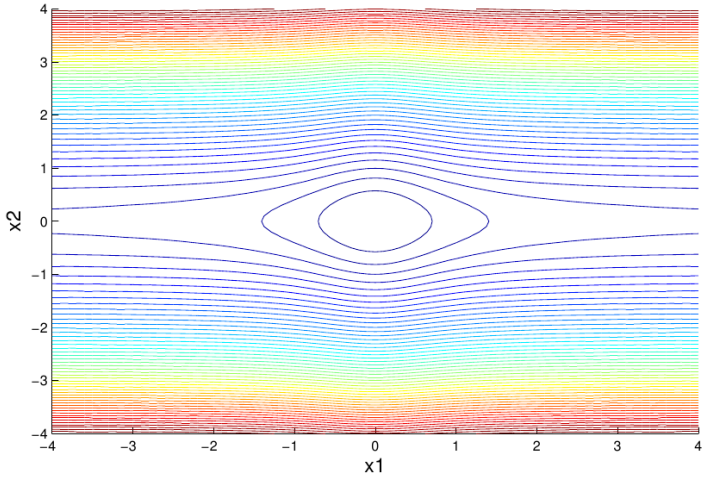
$$\dot{x_1} = -\frac{6x_1}{(1+x_1^2)^2} + 2x_2$$
$$\dot{x_2} = -\frac{2(x_1+x_2)}{(1+x_1^2)^2}$$

you are given the positive definite function $V(x) = \frac{x_1^2}{1+x_1^2} + x_2^2$ and told that, for this system, $\dot{V}$ is negative definite over the entire space. Is V a valid Lyapunov function which proves global asymptotic stability to the origin for the system? (Hint: Try simulating a few

trajectories of this system or plotting a few level sets of V to build more intuition before answering this problem).

- Yes
- No ✔

Explanation



The function $V(x)$ is not radially unbounded. This results in the level sets of V being unbounded (see the figure above for a picture of the level sets). Thus, even though V decreases along trajectories of the system, the system does not go to the origin.

(c) Suppose $V(x)$ is a valid Lyapunov function that proves global asymptotic stability of a system to the origin. Is it true that $(V(x))^2$ is also a valid Lyapunov function?

- Yes ✔
- No

Explanation

This is true since if $V(x)$ is positive definite, then so is $V^2(x)$. Further, the time-derivative of $V^2(x)$ is $2V\dot{V}$, which is negative definite because $\dot{V}$ is negative definite.

(d) Suppose $V(x)$ is a valid Lyapunov function that proves global asymptotic stability of a system to the origin. Is it true that $\tanh(V(x))$ is also a valid Lyapunov function?

- Yes
- No ✔

Explanation
The function $\tanh(V(x))$ is not radially unbounded and is thus not a valid Lyapunov function.

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Finding Lyapunov Functions

Finding Lyapunov Functions

0.0/15.0 points (graded)

Consider the system given by

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

Find a Lyapunov function $V(x_1, x_2)$ for this system in order to prove global asymptotic stability to the origin. Type in your answer in the MATLAB window below.

```
1 syms x1 x2 real;
2 V = ; % Type in your answer here in terms of x1 and x2
3
```

Unanswered

```
% For this question, the easiest way to obtain a Lyapunov function is to guess.
% The simplest possible Lyapunov function one could guess is V(x) = x1^2 + x2^2.
% Since this function is positive definite, we only need to check that its derivative
% is negative definite. The time derivative is -2*x1^4 - 2*x2^4, which is clearly negative defir

syms x1 x2 real;
V = x1^2 + x2^2;
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Sums of Squares (SOS)

Sums of Squares (SOS)

0.0/10.0 points (graded)

Consider the polynomial:

$$p(x_1, x_2) = 2x_1^4 + 2x_1^3 x_2 - x_1^2 x_2^2 + 5x_2^4.$$

Prove that this polynomial is nonnegative by finding a representation as follows:

$$p(x_1, x_2) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}^T Q \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix},$$

where Q is given by:

$$Q = \begin{bmatrix} 2, & a, & 1 \\ a, & 5, & 0 \\ 1, & 0, & b \end{bmatrix}.$$

Here Q must be positive semidefinite. Type in your values for a and b below. **Make sure that the resulting Q is positive semidefinite**.

```
1 a = ;
2 b = ;
3
```

Unanswered

```
% First, we denote v = [x1^2;x2^2;x1x2]. Then, we have:
% v'Qv =  2*x1^3*x2 + 2*x1^4 + 5*x2^4 + 2*a*x1^2*x2^2 + b*x1^2*x2^2.
% Since we want v'Qv to equal p(x1,x2), we must have 2a + b = -1 and
% hence, b = -1 - 2a. Thus, we can write Q as:
% Q = [2 a 1;
%      a 5 0;
%      1 0 (-1-2a)];
% Now, all we need to do is choose an a such that Q is positive semidefinite.
% One such choice is a = -1, b = 1

a = -1;
b = 1;
```

Run Code

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

Acrobot Region of Attraction

Acrobot Region of Attraction (Part I)

0.0/5.0 points (graded)

Prerequisites for this problem: You will need a semidefinite program solver. For this, you can either use SeDuMi (found here) or MOSEK (found here). MOSEK is typically a lot faster than SeDuMi and we recommend installing it. If you are using a precompiled version of Drake, you can install the solver of your choice from the above links. If you built drake from source, you can just do

```
cd /path/to/drake-distro
make options
```

and then set WITH_MOSEK or WITH_SEDUMI to ON. Then hit "c to configure", "g to generate and exit", and finally, from the drake-distro folder, do:

```
make
```

In this problem, we will use Drake to approximate the region of attraction (ROA) for a LQR controller that makes the Acrobot balance at the upright configuration. We have provided most of the code you'll need to compute ROAs using Drake in **acrobotROA.m** (found here). Please download this file and **place it in the examples/Acrobot folder** in Drake.

The code we have provided first sets up the Acrobot model and computes a LQR controller to balance the robot at the upright position. It also calls the Drake function **regionOfAttraction.m** to compute approximations of the ROA of the resulting closed-loop system. Please try to familiarize yourself with the different functions in regionOfAttraction.m and make sure you understand (at least at a high level) what they do. The regionOfAttraction.m code uses Sums-of-Squares (SOS) Programming to approximate ROAs. Your only task is to fill in the lines in the block of code that says "FILL THIS IN" (see below for instructions on this) in acrobotROA.m.

(a) Set the degree of the Lyapunov function to 2 and the degree of the Lagrange multipliers to 2 also. Next, use the "sampling" method to approximate the ROA. This option fixes the Lyapunov function (the cost-to-go function from LQR in our case) and searches for the largest sub-level set of the function such that $\dot{V} < 0$ for all points in the sub-level set. This is done by random sampling rather than SOS programming. Type in the value of the level set found below (i.e., the last value of "rho" printed out).

Answer: 6.81

Explanation
We set options.degL1 = 2 and options.degV = 2. Setting options.method = 'sampling' and running the code gives us a rho of 6.81.

(b) Now use the binary search method to approximate the ROA. This option again fixes the Lyapunov function and searches for the largest sub-level set of the function such that $\dot{V} < 0$ for all points in the sub-level set. However, this search is now done using SOS programming. Please enter the value of the level set found below (i.e., the last value of "rho" printed out).

Answer: 6.124

Explanation
Setting options.method = 'binary' and running the code gives us a rho of 6.124.

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Acrobot Region of Attraction (Part II)

0.0/10.0 points (graded)
(c) Finally, we will use SOS programming to search for both the Lyapunov function and the sub-level set. This is done using the "bilinear" option. The final Lyapunov function is printed to the MATLAB workspace (the variable "V"). The 1 sub-level set (i.e., the set of points x such that $V(x) < 1$) is the estimated ROA.

Find a point x_0 such that $V(x_0) < 1.0$ (i.e., a point inside the verified ROA). Ideally, choose a point close to the boundary of the verified ROA. The easiest way to do this is probably to use the plot of the ROA generated by the code. This is a plot of a two-dimensional slice of the

four-dimensional ROA, with the angular velocity coordinates set to 0. In other words, if we have a point (θ_1, θ_2) inside the gray region of the plot, the state $[\theta_1, \theta_2, 0, 0]$ lies inside the verified ROA.

Simulate the system from the point you found (we have provided you code for this in acrobotROA.m). Check that the Lyapunov function starts off less that 1 and then decreases to almost 0 (the plot of the value of the Lyapunov function vs. time will appear in Figure 2). Type in the Lyapunov function **V_grade** that gets printed to the MATLAB window and the 100 x 4 matrix x_grade below. Note: You do not have to modify V_grade in any way (e.g., taking out parentheses, etc. or changing the names of variables). Simply copy it below.

```
syms q1 q2 v1 v2
V_grade = ;
x_grade = [   ];

```

Unanswered

```
% For this problem, we set options.method = 'bilinear'.
% A point that lies in the ROA is x0 = [2.938;0.45;0;0].
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

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:

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

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**.

```
1 syms x1 x2
2 g_X0 = ;
3
```

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**.

```
1 syms x1 x2
2 g_Xu = ;
3
```

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.

```
1 syms x1 x2
2 B = ;
3
```

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

Trajectory Optimization

Trajectory Optimization

0.0/20.0 points (graded)

Suppose we have a trajectory optimization program for the Cart-Pole (so $x \in \mathbb{R}^4$ and $u \in \mathbb{R}$). For this problem, we will examine both shooting and direct transcription methods examined in class. We will suppose that both methods use forward Euler integration, that is:

$$x_{k+1} = x_k + hf(x_k, u_k)$$

for some fixed time step h. The programs will both be created as follows:

- The initial state, x_0 is fixed (and not a decision parameter).
- The final state, x_{20} is constrained to be in some goal region, that is $f_g(x_{20}) \geq 0$, where f_g is scalar valued.
- There is an obstacle which the states cannot penetrate, that is, $f_o(x_k) \geq 0$, where f_o is scalar valued.
- It is possible that the goal and obstacle regions have a non-empty intersection
- The final time and time step h are fixed.
- There are no torque limits, but the total cost is $\sum_k u_k^2$.
- Numbers of "decision variables" are counted as scalars. So if x_{10} were included in the decision variables, it would count as four.

Decision Variables

(a) How many decision variables does the *shooting* approach have?

Answer: 20

(b) How many decision variables does the *direct transcription* approach have?

Answer: 100

Explanation
In the shooting approach, the decision variables are the control inputs only (20). In the direct transcription approach, the state variables are also decision variables. The problem states that x_0 is fixed and not to be included, so the answer is 20 + 80=100.

Constraints

(c) How many constraints does the *shooting* approach have? Convert vector-valued constraints to multiple scalar-valued constraints to count them.

Answer: 21

(d) How many constraints does the *direct transcription* approach have? Convert vector-valued constraints to multiple scalar-valued constraints to count them.

Answer: 101

Explanation
The constraints for the shooting approach are the obstacle constraints (1 per time step) and the final state constraint, so 21 total. For the direct approach, the dynamics also form a constraint at each time step, giving an additional 80 constraints, for a total of 101.

Gradient Sparsity

Consider the constraint related to the goal region, $f_g(x_{20}) \geq 0$. For most numerical optimization approaches, it is important to calculate the gradient of the constraints and cost with respect to the decision variables. That is, if z is the list of all decision variables, we need to compute $\frac{\partial f_g(x_{20})}{\partial z}$, which will be a vector of length $\dim(z)$.

(e) For the *shooting* approach, how many non-zero entries are there in this gradient vector?

Answer: 20

(f) For the *direct transcription* approach, how many non-zero entries are there in this gradient vector?

Answer: 4

Explanation

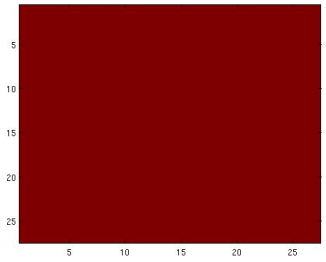
For the shooting approach, x_{20} depends on all of the control inputs u_k, since it is found by forward simulation. That means all 20 entries in the gradient vector will be non-zero. In the direct transcription approach, x_{20} is a decision variable, so $f_g\left(x_{20}\right)$ will only depend on the 4 variables that compose x_{20}.

Gradient Structure

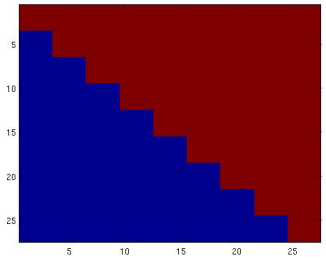
Suppose we write the gradient of all constraints as a matrix G, where the jth element of the ith row is $G_{i,j} = \frac{\partial h_i}{\partial z_j}$ where h_i is the ith constraint. Order both the z vector and the list of constraints h_i by the time index k which they most directly relate to. The structure of G greatly affects the efficiency and accuracy of various optimization approaches. The images below each option are graphical representations of the sparsity patterns, where red indicates the non-zero entries and blue the zero entries.

(g) For the *shooting* approach, what, if any, structure will G have? Note that G will not generally be square, so we use some of these terms loosely here. In particular, the "diagonal" here refers to the elements of G corresponding to the variables and constraints directly related to the same time index.

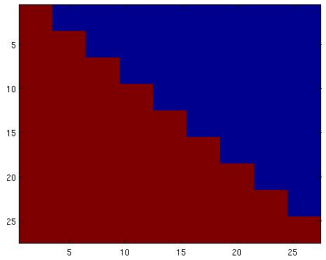
- Dense: many non-zero entries with no particular structure



- Upper Triangular: entries below the "diagonal" are all (or nearly all) zero

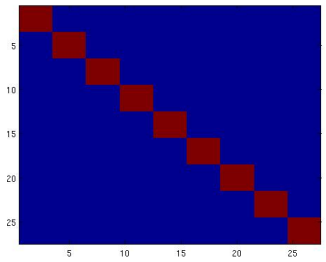


- Lower Triangular: entries above the "diagonal" are all (or nearly all) zero



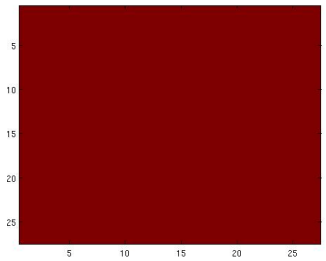
✔

- Block Diagonal: entries away from the diagonal are all (or nearly all) zero

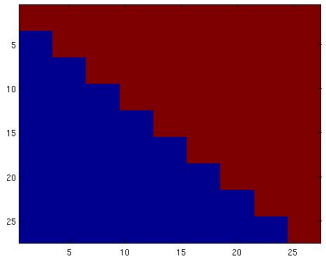


(h) For the *direct transcription* approach, what, if any, structure will G have?

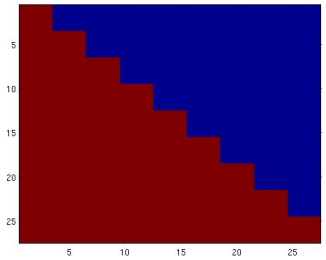
- Dense: many non-zero entries with no particular structure



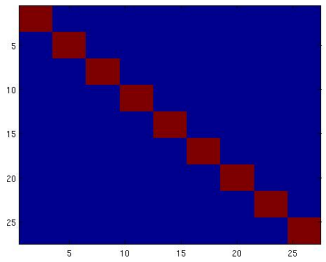
- Upper Triangular: entries below the "diagonal" are all (or nearly all) zero



Lower Triangular: entries above the "diagonal" are all (or nearly all) zero



Block Diagonal: entries away from the diagonal are all (or nearly all) zero



Explanation

For the shooting approach, constraints at time k will generally depend on all previous inputs, u_i for $i \leq k$, thus the lower triangular structure. In the direct method, constraints at time k generally only depend on nearby terms, like u_{k-1}, u_k, u_{k+1} and x_{k-1}, x_k, x_{k+1}, thus the block diagonal structure.

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Direct Collocation

Direct Collocation, Part 1

0.0/10.0 points (graded)

A popular and effective implementation of direct trajectory optimization is direct collocation. See the paper by Hargraves and Paris, linked in the Syllabus, for a reference. This approach defines the trajectory $x(t)$ as a *spline*, specified by its value at a series of knot points, and then enforces the constraint that the time derivative of this spline match the dynamics.

Let x_k and u_k be decision variables corresponding to knot points, where h is the time between knot points. Then, the spline of interest is defined as follows:

- For every k, $x(t)$ is defined to be a cubic polynomial for t in the interval t_k to t_{k+1}.
- $x(t_k) = x_k$ and $x(t_{k+1}) = x_{k+1}$
- $\dot x(t_k) = f(x_k,u_k)$ and $\dot x(t_{k+1}) = f(x_{k+1},u_{k+1})$

Let $t_0=0$, $f_0=f(x_0,u_0)$, and $f_1 = f(x_1,u_1)$. For t in the interval 0 to h, write the cubic polynomial for the spline $x_s(t)$. HINT: the cubic term is $((\frac{2}{h^3}(x_0-x_1) + \frac{1}{h^2}(f_0+f_1))t^3$.

Answer: x_0 + f_0*t + (3/h^2*(x_1-x_0) - 1/h*(2*f_0 + f_1))*t^2 + (2/h^3*(x_0-x_1) + 1/h^2*(f_0+f_1))*t^3

\(\)

Explanation

First, by evaluating $x_s(0)$ and $\dot x_s(0)$, we see that the constant and linear terms of the cubic are simply $x_0 + f_0t$. Now, evaluating $x_s(h)$ and $\dot x_s(h)$ gives two equations and one unknown (the quadratic term, since the cubic term is given in the hint). Either of these two can be solved for the quadratic term to get the answer.

Now, we extract the state and its derivative at the midpoint of each spline. Continuing the example above, find $x_c = x_s(.5h)$ and $\dot x_c = \left. \frac{dx_s}{dt}\right|_{t=.5h}$, where "c" stands for the collocation point.

$x_c=$

Answer: .5*(x_0+x_1) + h/8*(f_0 - f_1)

$$

$\dot x_c=$

Answer: -1.5*(x_0-x_1)/h - .25*(f_0+f_1)

$$

Thus far, we have computed a number of expressions in terms of the decision variables, but we have not yet written the constraints for the optimization program. If we let assume a first-order hold on control input, $u_c=.5(u_0+u_1)$, constrain the slope of the spline $\dot x_c$ to match the dynamics, evaluated at the collocation point:

$$\dot x_c - f(x_c,u_c) = 0$$

where we have one such constraint between every two knot points (so there is one fewer constraint than knot points).

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Direct Collocation, Part 2

0.0/10.0 points (graded)

To get some experience coding trajectory optimization algorithms, write the function that evaluates the constraint function and its gradient. We will use the pendulum as an example, with masses and gravity set to simplify the dynamics so $\ddot \theta + 10 \sin \theta + b\dot \theta = u$.

- This example will use 10 knot points, with the vector of decision variables: \begin{equation*} z = \begin{bmatrix} \theta_1 \\ \dot \theta_1 \\ \theta_2 \\ \dot \theta_2

\\ \vdots \\ \dot \theta_{10}\\u_1 \\ \vdots \\u_{10} \end{bmatrix} \end{equation*}

- Evaluate the constraint function \(g(z)\) and \(dg(z) = \frac{\partial g}{\partial z}\), where \(g\) is a \(18 \times 1\) vector and \(dg\) is a \(18 \times 30\) matrix.
- Order \(g(z)\) so that \begin{equation*} g(z) = \begin{bmatrix} \dot x_c - f(x_c,u_c) = 0 \text{ between knot points 1 and 2} \\ \dot x_c - f(x_c,u_c) = 0 \text{ between knot points 2 and 3} \\ \vdots\\ \dot x_c - f(x_c,u_c) = 0 \text{ between knot points 9 and 10} \\ \end{bmatrix} \end{equation*} where \(x\) has the usual ordering \(x = \begin{bmatrix} \theta \\ \dot \theta \end{bmatrix}\)
- You may find it useful to numerically confirm your computation of the gradient, or even use numerical differencing to calculate it in your solution.

```
h = .1;
b = .1+rand;
% decision variables
z = 10*randn(30,1);

% x is 2x10, where x(:,k) is [theta_k;\dot \theta_k]
x = reshape(z(1:20),2,[]);

% u is 1x10, where u(k) is u_k
u = reshape(z(21:30),1,[])';

% your code here
g =
dg =

```

Unanswered

```
g = zeros(18,1);
dg = zeros(18,30);
for k=1:9,
  x_0 = x(:,k);
  x_1 = x(:,k+1);
  u_0 = u(k);
  u_1 = u(k+1);

  f_0 = [x_0(2); u_0 - b*x_0(2) - 10*sin(x_0(1))];
  f_1 = [x_1(2); u_1 - b*x_1(2) - 10*sin(x_1(1))];

  % compute gradients as we go along
  % use the format [d/dx_0 d/dx_1 d/du_0 d/du_1]
  df_0 = [0 1 0 0 0 0;-10*cos(x_0(1)) -b 0 0 1 0];
  df_1 = [0 0 0 1 0 0;0 0 -10*cos(x_1(1)) -b 0 1];

  %collocation point and gradient in the same format
  x_c = .5*(x_0 + x_1) + h/8*(f_0 - f_1);
  dx_c = [.5*eye(2) .5*eye(2) zeros(2)] + h/8*(df_0 - df_1);

  %control u_c and gradient
  u_c = (u_0 + u_1)/2;
  du_c = [0 0 0 0 .5 .5];

  %using the answer from the previous part, slope of the spline and its gradient
  xdot_c = -3/2/h*(x_0-x_1) - 1/4*(f_0 + f_1);
  dxdot_c = -3/2/h*[eye(2) -eye(2) zeros(2)] - 1/4*(df_0 + df_1);

  %calculate pendulum dynamics at the collocation point and gradient
  f_c = [x_c(2); u_c - b*x_c(2) - 10*sin(x_c(1))];
  df_c = [dx_c(2,:); du_c - b*dx_c(2,:) - 10*dx_c(1,:)*cos(x_c(1))];

  % add to the g vector, and get the indexing right in dg
  g((1:2) + (k-1)*2) = xdot_c - f_c;
  dg((1:2) + (k-1)*2,[(1:4) + (k-1)*2,20 + (1:2) + (k-1)*1]) = dxdot_c - df_c;
end
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Catch the Falling Ball

Catch the Falling Ball

0.0/25.0 points (graded)

For this problem, we will make use of Drake to make an Acrobot "catch" a falling ball. In this case, catching means matching the position of the end of the Acrobot with the position of the ball. Imagine that the Acrobot has a big net attached to the end, if you'd like.

The state of the combined Acrobot-ball system is $\boldsymbol{x} = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}$ where

$$q = \begin{bmatrix} \theta_1 \\ \theta_2 \\ x_{ball} \\ z_{ball} \\ \phi_{ball} \end{bmatrix}$$

With θ_1 and θ_2 the usual angles of the acrobot, and the other three coordinates relating to the ball position and orientation. Note, for the entire problem, since we are only concerned with the position of the ball, you may safely ignore its orientation ϕ_{ball}.

We have provided stub code to introduce you to the way Drake constructs trajectory optimization problems, available as a .zip here. Please unzip the stub code into the **examples/Acrobot** directory. The code is set to use the Direct Collocation method that you explored earlier in this problem set. The initial optimization problem to solve is to minimize torque squared (see the *running_cost_fun* in the code), while matching the final positions described above. The Acrobot and ball are given some initial state, which will bring the ballistic trajectory of the ball close enough to catch. You will note that the duration of the trajectory is restricted to be $\mathbf{1.5} \leq \boldsymbol{T} \leq \mathbf{3}$.

NOTICE: when submitting responses to this question, **x_grade** is a fairly large matrix. Make sure that when you copy and paste it into the boxes below, that it pasted properly. At the end of the sample code, it prints out **x_grade** in a format that should help facilitate this copy-paste. Also see Matlab's *clipboard* function.

Part 1

The first task is to get *pset05_catch.m* working. You will find a few places in the file that indicate "YOUR CODE HERE." In particular, you need to finish the constraint on the final state and choose the initial state and control vectors for the optimization program. For this part, you do not yet need to finish the function *final_state_obj*.

A successful implementation will find a locally optimal trajectory, where the Acrobot catches the ball! Execute the script

```
run_pset5_catch_parts_1_and_2.m
```

And submit **x_grade** below. Make sure that the trajectory optimization returned a feasible solution (that there was no error)! You can paste the output of

```
mat2str(x_grade)
```

in the box, or you can use this matlab command:

```
clipboard('copy', x_grade)
```

to copy the value of x_grade to your clipboard, and then paste that result directly into the box.

```
1
```

Press ESC then TAB or click outside of the code editor to exit

Unanswered

```
% intialization can play a significant role in convergence of nonlinear programs,
% as you hopefully found out by exploration in this problem.
%
% one choice is to use a random iniitalization for u, and to use a constant x
trajectory, such as
x_init_vec = repmat(x0,1,N);
u_init_vec = randn(1,N);

% the "catching" constraint can be formulated as
q = x(1:5);
qd = x(6:10);
kinsol = obj.doKinematics(q);

hand_body = 3;% body index, so p.body(3) is the lower link
pos_on_hand_body = [0;-2.1]; % position of the "hand" on the lower link, 2.1m is the
length

[hand_pos,dHand_pos] = obj.forwardKin(kinsol,hand_body,pos_on_hand_body);

ball_pos = [q(3);q(4)];
dBall_pos = [zeros(2,2) eye(2) zeros(2,1)];

% note the 2x5 zero matrix at the end of df, because f is independent of qdot
f = ball_pos - hand_pos;
df = [dBall_pos - dHand_pos zeros(2,5)];
```

Part 2

Now, we would like to add the additional cost that awards catching the ball at a higher point in its trajectory. There are a number of different ways to do this, and the precise implementation is left to you. See the stub function *final_state_obj* for instructions on how to create this cost within Drake. Ensure that the cost is significant enough that it affects the outcome, since it is being balanced with the original torque squared cost. Again, execute the script

```
run_pset5_catch_parts_1_and_2.m
```

And submit **x_grade** below. Your solution must catch the ball at least slightly higher than the nominal, 3 second motion.

```
1
```

Press ESC then TAB or click outside of the code editor to exit

Unanswered

```
% There were many ways to reward an early catch, and the grader was only looking for
the time of the
% catch to occur 10ms earlier than it did in Part 1. Some submissions to this
problem failed to
% satisfy the catch criteria at all, and one possibility for this is that the
submission resulted
% from a failure of the solver. Poorly conditioned objective functions and
constraints, as well as bad
% initial seeds can cause this sort of behavior, and it's important to check any
errors/warnings
% Here's one possible objective function, which explicitly rewards the final height
of the hand.
q = x(1:5);
qd = x(6:10);
kinsol = obj.doKinematics(q);

hand_body = 3;% body index, so p.body(3) is the lower link
pos_on_hand_body = [0;-2.1]; % position of the "hand" on the lower link, 2.1m is the
length

[hand_pos,dHand_pos] = obj.forwardKin(kinsol,hand_body,pos_on_hand_body);

f = -100*hand_pos(2);
df = -100*[0 dHand_pos(2,:) zeros(1,5)];
```

Part 3

For the final part of this problem, implement a time-varying LQR controller. In particular, we would like to choose a cost that penalizes the distance from the ball to the hand at the point where the Acrobot should be catching the ball. Since the trajectory was designed with this as a constraint, it should not be too hard to generate an appropriate Q_f. In *run_pset5_part3*, set

the LQR cost terms. The script will use the time-varying controller to simulate a couple of times with perturbed initial conditions. Verify that your controller finishes close to the ball (within 0.07 m), and submit **x_grade** below.

HINT: Try finding a Q_f such that $x(T)^T Q_f x(T)$ approximates the distance-squared between the hand and the ball! This is a little tricky, simply guessing Q_f is unlikely to work.

```
1
```

Press ESC then TAB or click outside of the code editor to exit

Unanswered

```
% The key here was to think carefully about Qf, rather than choosing something
simple like the
% identity matrix To directly penalize the final distance from the hand to the ball,
consider
% the function f = (distance ball to hand)^2. If xf is the final state from the
nominal trajectory,
% then f(xf) = 0 and df/dx(xf) = 0.
%
% Then, if H is the hessian of f, then (x-xf)^T H (x - xf) is the quadratic
approximation of f around
% the nominal.  Since H isn't positive definite, we might want to add some
regularization, although it
% is not strictly necessary. One solution is to take Qf = K*H + I, code shown below
for K=100.

% get final state
xf = xtraj.eval(3);
qf = xf(1:5);

% same code as in the constraint function
kinsol = p.doKinematics(qf);

hand_body = 3;% body index, so p.body(3) is the lower link
pos_on_hand_body = [0;-2.1]; % position of the "hand" on the lower link, 2.1m is the
length

[hand_pos,dHand_pos] = p.forwardKin(kinsol,hand_body,pos_on_hand_body);

ball_pos = [qf(3);qf(4)];
dBall_pos = [zeros(2,2) eye(2) zeros(2,1)];

f = ball_pos - hand_pos;
df = [dBall_pos - dHand_pos zeros(2,5)];

% The Hessian is actually df'*df!
Q = eye(10);
R = 1;
Qf = Q + df'*df*100;
```

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

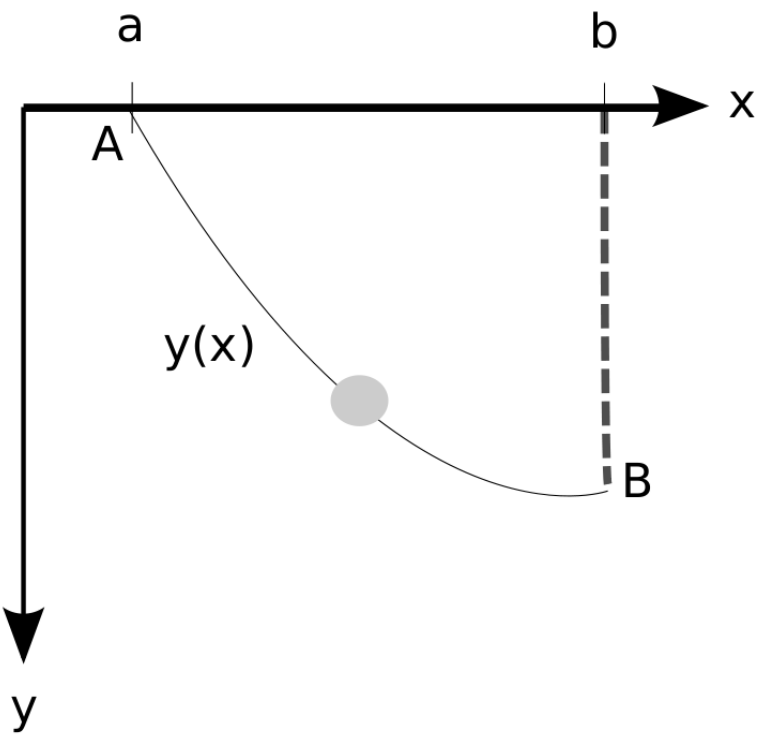
Brachistochrone

The Brachistochrone Problem, Part 1

0.0/10.0 points (ungraded)

In 1696, Johann Bernoulli challenged his fellow mathematicians to solve the "problem of the brachistochrone". The story goes that Newton stayed up all night and had the correct solution by the morning. Hopefully, you'll be quite a bit quicker than he was!

In this question, we will formulate this problem as a minimum-time optimal control problem.



Given two points A and B as shown in the figure above, the goal is to find a curve $y(x)$ joining A and B such that a body constrained to move along this curve under the influence of gravity goes from A to B in the least possible time. Think of the curve as a wire and the body as a bead that moves along the wire. We will make the following assumptions:

- There is no friction between the bead and the wire.

- The point A is higher than B
- The bead is released at rest from A
- The potential energy of the bead is 0 at the point A.

Let $u(x)$, which is our equivalent to "control" for this problem, be $\frac{dy}{dx}$. Then, the arclength of the curve from A to B is $\int_a^b \sqrt{1+u(x)^2}dx$

(a) Compute the speed of the bead $v(y)$ as a function of y. Your expression should be in terms of the gravitational acceleration g (in addition to y). Type in your expression below. (Hint: Use the fact that energy is conserved).

Explanation
Since energy is conserved, we know $0 = .5mv^2 - mgy$, so v = \sqrt\{2gy} .

Answer: sqrt(2*g*y)

(b) We can use the answer from the previous part to write down the brachistochrone problem as a minimum-time optimal control problem where the "control input" is $u(x)$, and the cost is of the form:

$$J(u(x), y(x)) = \int_a^b g(u(x), y(x))\, dx.$$

Write down the expression for $g(u, y)$ below. (Again, please abbreviate $y(x)$ and $u(x)$ to y and u respectively. Thus, your answer should be in terms of the symbols u, y, and g).

Answer: sqrt((1+u^2)/(2*g*y))

Explanation
Total time here can be written as the integral of the differential length over the speed, integrated along the curve, so $T = \int_a^b \sqrt{\frac{1+u^2}{2gy}}$

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

The Brachistochrone Problem, Part 2

0.0/15.0 points (ungraded)

For this final part, we will use Pontryagin's Minimum Principle to derive the differential equations that describe the solution.

Again, thinking of this as an optimal control problem, we introduce the costate variable $\lambda(x)$ and write the Hamiltonian

$$H(y(x), u(x), \lambda(x), x) = \lambda(x)\, u(x) + g(u(x), y(x))$$

(c) Pontryagin's Minimum princple states that $u(x)$ minimizes the Hamiltonian for all $y(x), \lambda(x), x$. Use this to solve for λ in terms of the symbols $u, x, g,$ and y.

λ=

Answer: -u/sqrt(2*g*y*(1+u^2))

Explanation

Taking the derivative $0 = \frac{\partial H}{\partial u} = \lambda + \frac{\partial g}{\partial u} = \lambda + \frac{u}{\sqrt{(1+u^2)2gy}}$, we can easily find λ as given in the answer.

(d)Again from Pontryagin, we know that the costate dynamics are $\lambda' = -\frac{\partial g(u,y)}{\partial y}$ Find the dynamics of the Hamiltonian, $H' = \frac{dH}{dx}$. Again, write your answer in terms of the symbols $u, x, g,$ and y. HINT: Your answer will not depend on $u' = \frac{du}{dx}$!

Answer: 0

Explanation

Expand the derivative H' and substitute:

$$\begin{aligned} H' &= \frac{\partial H}{\partial y}\frac{dy}{dx} + \frac{\partial H}{\partial u}\frac{du}{dx} + \frac{\partial H}{\partial \lambda}\frac{d\lambda}{dx} + \frac{\partial H}{\partial x} \\ &= \frac{\partial g}{\partial y}y' + 0 + u\lambda' + 0 \\ &= \frac{\partial g}{\partial y}u - u\frac{\partial g}{\partial y} \\ &= 0 \end{aligned}$$

(e) Let $k = H\left(y\left(a\right), u\left(a\right), \lambda\left(a\right), a\right)$ be the initial value of the Hamiltonian. Use the results above to find a governing equation of motion for the Brachistochrone. Your answer should be of the form

$$1 = \left(\alpha + u^2\right)\beta$$

Find α and β in terms of $x, g, y,$ and k.

$\alpha =$

Answer: 1

$\beta =$

Answer: 2*g*y*k^2

You should confirm that there is a parametric solution solution to this differential equation of the form (called a *cycloid*)

$$\begin{aligned} x\left(s\right) &= \frac{1}{4gk^2}\left(s - \sin s\right) \\ y\left(s\right) &= \frac{1}{4gk^2}\left(1 - \cos s\right) \end{aligned}$$

Explanation
From the previous part, we know that H is constant along the trajectory, so we can write

$$H\left(y\left(x\right),u\left(x\right),\lambda\left(x\right),x\right)=\lambda\left(x\right)u\left(x\right)+g\left(u\left(x\right),y\left(x\right)\right)$$

$$k=-\frac{u}{\sqrt{\left(1+u^2\right)2gy}}u+\sqrt{\frac{1+u^2}{2gy}}$$

$$k=-\frac{u^2}{\sqrt{\left(1+u^2\right)2gy}}+\frac{1+u^2}{\sqrt{\left(1+u^2\right)2gy}}$$

$$k=\frac{1}{\sqrt{\left(1+u^2\right)2gy}}$$

$$1=\left(1+u^2\right)2gyk^2$$

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

Limit Cycles

Limit Cycles

0.0/20.0 points (graded)

This question will test your understanding of limit cycles and Poincare analysis with a series of short questions.

(a) Which of the following statements about Poincare analysis are true? Select all that apply.

- [ ] A Poincare section is always two dimensional.
- [ ] The Poincare section must be perpendicular to the periodic orbit under consideration.
- [ ] A periodic orbit is (locally orbitally) stable if the Poincare return map is (locally) asymptotically stable. ✔
- [ ] The Poincare return map is (locally) asymptotically stable if the periodic orbit is (locally orbitally) stable. ✔
- [ ] The linearization of the Poincare return map always has one eigenvalue equal to 0. ✔

Explanation

The first choice is incorrect since a Poincare section is $n-1$ dimensional in general. The second choice is incorrect since a Poincare section need only be transverse to the periodic orbit (and not necessarily perpendicular). The third and fourth choices are correct (as discussed in class). The last choice is also correct since perturbations *along* the limit cycle lead to no change in the point at which the trajectory returns to the Poincare section. Hence, perturbations along the trajectory correspond to a 0 eigenvalue of the linearization of the Poincare map.

(b) Which of the following is a limit cycle? Select all that apply.

- [ ] The unstable fixed point corresponding to the pendulum in the upright configuration.
- [x] The periodic orbit of the Van der Pol oscillator. ✔
- [ ] A periodic orbit of the undamped pendulum, e.g., the periodic orbit with constant energy $E > 0$.
- [ ] The homoclinic orbit of the undamped pendulum.
- [x] The periodic orbit corresponding to the rimless wheel rolling down a gentle ramp. ✔

Explanation
A limit cycle is a (stable or unstable, but not marginally stable) periodic orbit, where the period must be a real number that is *strictly* greater than 0. Thus, a fixed point is not a limit cycle (since the period is not strictly greater than 0). The periodic orbit of the Van der Pol oscillator is a limit cycle, as discussed in class. Both the homoclinic orbit and a periodic orbit of the undampled pendulum are not limit cycles since they are marginally stable. The rimless wheel's periodic orbit is a stable periodic orbit, and hence a limit cycle.

(c) Suppose we have a system given by:

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

with $x \in \mathbb{R}$, i.e., a one dimensional system. Is it possible for this system to have a limit cycle?

- [ ] Yes
- [x] No ✔

Explanation
One dimensional (non time-varying) systems cannot have limit cycles. This is easy to see. Suppose we did have a limit cycle that started off at x_0, went to x_1 and returned to x_0 (here $x_0 < x_1$). Let x_m be a point between x_0 and x_1. Then the vector field at x_0 points to the

right when the system is moving from x_0 to x_1 and points to the left when the system is going back from x_1 to x_0. Since the system is not time-varying, this is impossible and we have a contradiction.

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Non-existence of Limit Cycles

Non-existence of limit cycles (Part A)

0.0/10.0 points (graded)

A **gradient system** is a system of the following form:

$$\dot{x} = -\nabla V(x) \quad (1),$$

for some twice continuously differentiable scalar-valued function $V(x)$.

It is straight-forward to prove that gradient systems don't have any periodic orbits (we are not counting fixed-points as periodic orbits here, and we assume that $V(x)$ is not the 0 function). We can use a proof by contradiction in order to show this. Suppose we had a periodic orbit. Consider the change in V after one circuit around the periodic orbit. Denote this change as ΔV. For a periodic orbit, we must have $\Delta V = 0$. However, we also have:

$$\Delta V = \int_0^T \dot{V}(x(t))\,dt = \int_0^T \nabla V(x(t)) \cdot (-\nabla V(x(t)))\,dt = \int_0^T -\|\nabla V(x(t))\|^2 dt < 0.$$

This is a contradiction.

(a) Which of the following are gradient systems?

- [x] All linear systems $\dot{x} = Ax$ with $A = A^T$. ✔
- [ ] All linear systems $\dot{x} = Ax$.
- [x] All one-dimensional systems $\dot{x} = f(x)$, where $f(x)$ is continuously differentiable. ✔
- [ ] All globally asymptotically stable systems.

Generating Speech Output

All linear systems of the form $\dot{x} = Ax$ with $A = A^T$ are gradient systems with $V = 0.5x^T Ax$. Linear systems with $A \neq A^T$ are not generally gradient systems. To see this, consider the two dimensional system:

$$\dot{x}_1 = ax_1 + bx_2 := f(x_1, x_2)$$
$$\dot{x}_2 = cx_1 + dx_2 := g(x_1, x_2).$$

Recall from your multi-variate calculus course that a two-dimensional system is a gradient system if and only if the following holds:

$$\frac{\partial}{\partial x_2} f(x_1, x_2) = \frac{\partial}{\partial x_1} g(x_1, x_2).$$

Now, if $b \neq c$, this condition will not hold.
All one-dimensional systems $\dot{x} = f(x)$ are gradient systems since we can set $V(x) = -\int f(x)\, dx$. Globally asymptotically stable systems are not generally gradient systems. Consider a stable linear system $\dot{x} = Ax$ with $A \neq A^T$ for example.

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Non-existence of limit cycles (Part B)

0.0/10.0 points (graded)
(b) Consider the system given by

$$\dot{x} = y + 2xy$$
$$\dot{y} = x + x^2 - y^2.$$

Prove that this system does not have a periodic orbit by finding a function $V(x, y)$ such that the system is of the form (1) above. Type in $V(x, y)$ below (make sure it has the correct sign!).

```
1 syms x y real;
2 V = ; % Type in your answer here in terms of x and y
3
```

Generating Speech Output

Unanswered

```
syms x y real;

% As is easily verified by taking gradients, one possible V is:
V = -x*y - x^2*y + (1/3)*y^3
```

Run Code

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

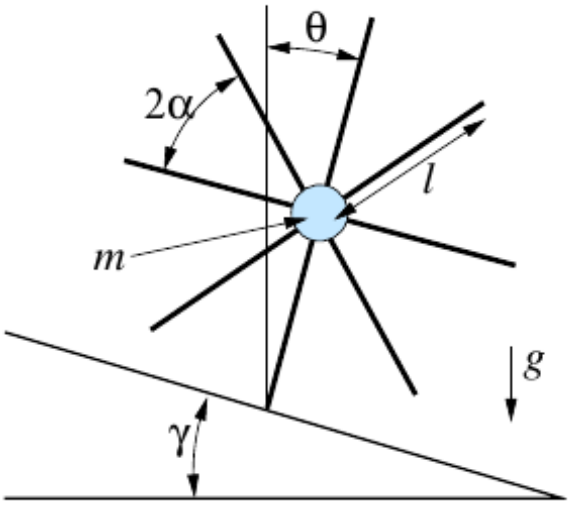
Generating Speech Output

Rimless Wheel

The Rimless Wheel (Passive Dynamics I)

0.0/8.0 points (graded)

For this problem, you should be able to compute all of the quantities in closed form (no numerical simulations should be necessary).



(a) If we assume that the wheel is started in a configuration directly after a transfer of support, then forward walking occurs (i.e., the system takes a step) when the system has an initial velocity $\dot{\theta}\left(0^{+}\right) > \omega_1$. When the next foot touches down the conversion of potential energy into kinetic energy yields the velocity before touchdown $\dot{\theta}\left(t^{-}\right)$. Derive the expressions for ω_1 and $\dot{\theta}\left(t^{-}\right)$ in terms of $g, l, \alpha, \gamma, \dot{\theta}\left(0^{+}\right)$. Type in your answers below. Please write α, γ, and $\dot{\theta}\left(0^{+}\right)$ as "alpha", "gamma" and "thdot0" respectively.

$\omega_1 =$

Answer: sqrt((2*g/l)*(1 - cos(alpha - gamma)))

Generating Speech Output

$\dot{\theta}\left(t^{-}\right) =$

Answer: sqrt(thdot0^2 + (4*g/l)*sin(alpha)*sin(gamma))

Explanation

Using conservation of energy, we get:

$$\frac{1}{2}ml^2\dot{\theta}^2\left(0^{+}\right) + 2mgl\sin\left(\alpha\right)\sin\left(\gamma\right) = \frac{1}{2}ml^2\dot{\theta}^2\left(t^{-}\right).$$

Then we get:

$$\dot{\theta}\left(t^{-}\right) = \sqrt{\dot{\theta}^2\left(0^{+}\right) + \frac{4g}{l}\sin\left(\alpha\right)\sin\left(\gamma\right)}.$$

Letting $\omega_1 = \dot{\theta}\left(0^{+}\right)$, when $\theta = 0$ and $\dot{\theta} = 0$, we have:

$$\frac{1}{2}ml^2\omega_1^2 = mgl\left(1 - \cos\left(\alpha - \gamma\right)\right).$$

Thus, we get:

$$\omega_1 = \sqrt{\frac{2g}{l}\left(1 - \cos\left(\alpha - \gamma\right)\right)}$$

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

The Rimless Wheel (Passive Dynamics II)

0.0/2.0 points (graded)

(b) Write down a condition of the form $\gamma > f\left(\alpha, g, l\right)$ such that ω_1 is non-existent when this condition holds. Type in your answer for $f\left(\alpha\, g, l\right)$ below (as before, write α as "alpha"\). Think about what this condition means physically.

Generating Speech Output

Answer: alpha

Explanation
When $\gamma = \alpha$, with an arbitrary small initial speed, the rimless wheel will always roll down by gaining potential energy. When $\gamma > \alpha$, the rimless wheel will roll down due to gravity even when it starts with zero velocity.

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

The Rimless Wheel (Gait Design)

0.0/7.0 points (graded)
(c) Consider a rimless wheel with $m = 1$ kg, $g = 9.8$ m/s^2, $\alpha = \frac{\pi}{8}$ rad, and $\gamma = 0.08$ rad. For what leg length l, is the steady-state rolling speed of the system (measured when the pendulum is vertical over the stance feet: $\theta = 0$) equal to 4.0 rad/s? **The tolerance for this answer will be 0.001.**

Answer: 0.0155

Explanation
In the steady-state rolling condition, $\dot{\theta}\left(t^-\right)\cos\left(2\alpha\right) = \dot{\theta}\left(0^+\right)$. Using the result from part (a),

$$\dot{\theta}\left(0^+\right) = \cot\left(2\alpha\right)\sqrt{\frac{4g}{l}\sin\left(\alpha\right)\sin\left(\gamma\right)}.$$

At $\theta = 0$, we get:

$$\dot{\theta} = \sqrt{\dot{\theta}^2\left(0^+\right) - \frac{2g}{l}\left(1 - \cos\left(\alpha - \gamma\right)\right)}$$

Then we get:

$$l = \frac{4g\sin\left(\alpha\right)\sin\left(\gamma\right)\cot^2\left(2\alpha\right) - 2g\left(1 - \cos\left(\alpha - \gamma\right)\right)}{\dot{\theta}^2} = 0.0155.$$

Generating Speech Output

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

The Rimless Wheel (Terrain Stability)

0.0/8.0 points (graded)

(d) Consider a rimless wheel with $m = 1$ kg, $l = 1$ m, $g = 9.8$ m/s^2, and $\alpha = \frac{\pi}{8}$ rad rolling down a slope, $\gamma = 0.16$ rad. Some time after reaching the rolling steady state, the terrain shallows to $\gamma = 0.02$ rad for a distance d, then returns to (and remains at) $\gamma = 0.08$ rad. You may assume that the shallow slope begins and ends precisely at a position of touchdown (not somewhere between steps). What is the largest value of d, subject to these constraints, for which the system will be at a rolling fixed point when $\gamma = 0.08$ as $t \to \infty$?
The tolerance for this answer will be 0.001.

Answer: 0.7654

Explanation

When $\gamma = 0.08$ rad, the steady state speed is 1.0949. Also, $\omega_1 = 0.9749$. When the rimless wheel starts with $\dot{\theta}\left(0^+\right) = \omega_1$, after the first collision, the velocity $\dot{\theta}\left(t^+\right) = 1.0366 > \omega_1$, so as long as the rimless wheel starts at a velocity greater than ω_1, it will finally reach the steady state rolling condition. When $\gamma = 0.02$ rad, the rimless wheel starts with $\dot{\theta} = 1.5460$, after the first collision, $\dot{\theta} = 1.1597 > \omega_1$, after the second collision, $\dot{\theta} = 0.9069 < \omega_1$, so only one collision is allowable and $d = 2l\sin\left(\alpha\right) = 0.7654$.

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

Generating Speech Output

Poincare Analysis (Compass Gait)

Poincare Analysis on Compass Gait (Part I)

0.0/10.0 points (graded)

In this problem we will investigate a limit cycle for the unactuated compass gait model. To see the compass gait model in action, navigate to drake/examples/CompassGait and call CompassGaitPlant.run(). Look around in CompassGaitPlant and try to understand how the hybrid system is set up using modes, guards, and transitions.

In the rest of this problem, keep in mind that the compass gait model is a hybrid system, which has both discrete state $x_d \in \mathbb{Z}$ (the mode that the system is in) and continuous state $x_c \in \mathbb{R}^4$ (the joint angles and velocities). The complete state vector is $x = [x_d; x_c]$.

First, write a function with the signature **xf = strideFunction(p, x0)**, where p is a CompassGaitPlant, x0 is the initial continuous state and xf is the continuous state right after a single step has been taken. In the function, you may assume that the initial discrete state is 1 and that no step will ever take longer than 1 second.

Hint 1: you can pass in the initial state of the system as the third argument to the simulate function.

Hint 2: the output of simulate will be a HybridTrajectory. The traj field of a HybridTrajectory is a cell array of PPTrajectories (piecewise polynomial trajectories) corresponding to the sequence of discrete modes that the system goes through. A PPTrajectory can be evaluated at a given time using the eval method.

Note: For this problem, please make sure to NOT round off your answers in intermediate steps. In other words, please keep around all the digits you computed in Matlab since numerical errors can propagate quickly and cause problems later down the line. To be safe, set your tolerances much tighter than the ones we list in the problems (since there is almost no computational cost to doing this).

(a) What is the output of strideFunction(p, [0; 0; 2; -0.4])? Type in your answer below (xf must

Generating Speech Output

tolerance for this answer is 0.001.

```
xf = ;

```

Unanswered

```
%% Stride function:
% function xf = strideFunction(p, x0)
% traj = simulate(p,[0 1], [1; x0]); % Simulate from x0 with mode = 1
% xf = traj.traj{2}.eval(traj.traj{2}.tspan(1)); % Get state right after collision
% end

% For [0;0;2;-0.4], we have:
xf = [-0.32600174
       0.22128199
      -0.38086431
      -1.0879922];
```

Run Code

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

Poincare Analysis on Compass Gait (Part II)

Generating Speech Output

(b) The state x = [1; 0; 0; 2; -0.4] is attracted to a stable limit cycle of the system. Use your strideFunction to find this stable limit cycle, identified by the continuous state x_0 just after a step has been taken. What is x_0? Type in your answer below (it must have size 4×1). **Our tolerance for this answer is 0.001.**

```
x0 = ;

```

Unanswered

Generating Speech Output

```
%% The fixed point can be found by repeatedly simulating the system using strideFunction as foll
%
%% find fixed point
% p = CompassGaitPlant();
% x0 = p.getInitialState();
% x0 = x0(2:end);
% epsilon = 1e-12;
% while true
%   x0_new = strideFunction(p, x0);
%   if norm(x0_new - x0) < epsilon
%     break;
%   end
%   x0 = x0_new;
% end
% disp('x0:');
% fprintf('%0.8g\n', x0');
% fprintf('\n');

% Then, we get:
x0 =  [-0.32338855
        0.21866879
       -0.37718213
       -1.0918269];
```

Run Code

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

Poincare Analysis on Compass Gait (Part III)

0.0/10.0 points (graded)

(c) Next, use your strideFunction to numerically find the monodromy matrix A. This is the linearization of the Poincare map at the point $\boldsymbol{x_0}$. Type in your answer below. **Our tolerance for this answer is 0.01.**

Generating Speech Output

```
2
```

Unanswered

```
%% The A matrix can be computed by numerical differentiation. So, for example, the first
%% column can be computed as follows:
% x1 = strideFunction(p,x0);
% x2 = strideFunction(p,x0+[1e-5;0;0;0]);
% A_1 = (x2-x1)/1e-5;

% Computing the other columns in a similar way, we get:
A = [0.3185    1.5933   -0.0259    0.5646;
    -0.3185   -1.5933    0.0259   -0.5646;
    -2.3383   -2.7989    0.0954   -1.3183;
    -0.1686    2.5803   -0.0633    0.9116];
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Poincare Analysis on Compass Gait (Part IV)

0.0/5.0 points (graded)

Generating Speech Output

(d) What are the Floquet multipliers (eigenvalues of the monodromy matrix) corresponding to the stable limit cycle? Type in your answer below as a 4×1 vector of complex numbers. Don't worry about ordering the multipliers in any particular order. **Our tolerance for this answer is 0.05 (in terms of the magnitude of each element in the vector).** (Check that the linearized Poincare map is stable, as expected).

```
1 f = ;
2
```

Unanswered

```
% Computing the eigenvalues of A using eig(A), we get:
f = [-0.1997 + 0.5445i;-0.1997 - 0.5445i;0.1316 + 0.0000i;-0.0+0.0000i];
% As expected, one of the eigenvalues is 0
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Generating Speech Output

Controlling the Compass Gait

Controlling the Compass Gait (Part I)

0.0/8.0 points (ungraded)

In this problem we will stabilize the passive limit cycle we obtained in the previous problem using a time-varying LQR (TVLQR) controller. The first step in order to do this is to modify the standard continuous-time TVLQR approach to work for hybrid systems. The Riccati equation associated with this problem is the so-called "Jump Riccati Equation" (JRE):

$$-\dot{S}(t) = Q - S(t)B(t)R^{-1}B(t)^T S(t) + S(t)A(a) + A(t)^T S(t) \quad (1)$$

$$S(t^-) = A_d^T S(t^+) A_d. \quad (2)$$

Here, equation (1) is the usual evolution of the cost (same as the continuous time case), which we solve backwards in time with a final-value constraint $S(t_f) = Q_f$. The second equation propagates the cost backwards through a hybrid event ("jump"). Here A_d is the linearization of the discrete transition dynamics (in our case, it is the linearization of the collision dynamics for the compass gait).

The second modification we need to the standard continuous-time TVLQR is to make its solution periodic. In other words, we want the cost-to-go at $t = 0$ to be the same as the cost-to-go at the final time $t = t_f$ (since the limit cycle is periodic). It is well-known (but not trivial to prove!) that repeatedly solving the equations (1) and (2) backwards in time will result in convergence to a periodic $S(t)$. Hence, we solve (1) backwards in time for the interval $[t^+, t_f]$ with a final-value constraint $S(t_f) = Q_f$. Then, we solve (2) using the $S(t^+)$ obtained from (1). We then solve (1) again with the final value constraint $S(t_f) = S(t^-)$, and so on until convergence.

We have provided stub code here that solves the continuous time equation (1) using Drake. Your task is to fill in the section corresponding to the discrete jump equation (2), and to check for convergence of $S(t)$ to a periodic solution. It might be helpful to take a look at the function **collisionDynamics** in CompassGaitPlant.m. The inputs to this function are the hybrid mode ("1" in our case), time, pre-impact state (a 4×1 vector) and control input. The first output is the post-impact state. The columns of the last output (dxp) are the gradients of

the post-impact state with respect to the different inputs of the function. So, the columns of interest to us are the columns 3 through 6 (since these are the gradients with respect to the pre-impact state).

Note: This problem relies on your answer(s) from the previous problem. To be safe, make sure to use all the digits you comptued in Matlab in the previous part. Also set your tolerances much tighter than the ones we list in the problems (since there is almost no computational cost to doing this).

(a) What is the value of Q_f when the solution has converged (the variable "Qf_converged" in the code)? **Our tolerance for this answer is 0.01.**

```
Qf_converged = ;

```

Unanswered

```
% The Qf_converged we compute is:
Qf_converged =   1.0e+03 * [6.490617561949686  -2.137322594070590 -0.122512533982038   1.1995577
  -2.137322594070590   0.732684500175522   0.039887204353096  -0.391443040435211;
  -0.122512533982038   0.039887204353096   0.002331333037853  -0.022717723218988;
   1.199557787078113  -0.391443040435211  -0.022717723218988   0.222366589377603];

% Our full code is shown below:

% Compass Gait plant
p = CompassGaitPlant();

%% Get limit cycle initial condition (FILL ME IN) %%%%%%%%%%%%%%%%%%%%%%%%%
x0 = [-0.32338855;
0.21866879;
-0.37718213;
-1.0918269]; % Your x0 from part (b) of the problem "Poincare analysis on compass gait"
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x0 = [  -0.323388544149323
   0.218668789029664
  -0.377182130914653
  -1.091826912735189
];

% Simulate forwards from x0
xf = strideFunction(p,x0); % Your strideFunction from the previous problem

% Check that the resulting trajectory is periodic
if max(abs(xf-x0)) > 1e-4
    error('Simulating x0 does not result in a limit cycle');
end

% Simulate forwards from x0 to get the trajectory
xtraj = simulate(p,[0 1], [1; x0]);
xtraj = xtraj.traj{1}; % Extract first part of the trajectory (before collision)
ts = xtraj.getBreaks(); % Get time samples of trajectory
utraj = PPTrajectory(spline(ts,zeros(1,length(ts)))); % Define nominal u(t),
                                                      % which is all zeros since our
                                                      % system was passive

% Set frames
xtraj = xtraj.setOutputFrame(p.modes{1}.getStateFrame);
utraj = utraj.setOutputFrame(p.modes{1}.getInputFrame);


%% Stabilize using tvlqr (FILL ME IN) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define Q, Qf, R
```

```
Q = diag([10 10 1 1]);
R = 1;
Qf = Q;

options = struct();

converged = false;

while ~converged
% tvlqr for continuous phase
[tv,V] = tvlqr(p.modes{1},xtraj,utraj,Q,R,Qf,options);
QfV = Qf;

% Jump equation (FILL ME IN)
S_t_plus = V.S.eval(0);
xend = xtraj.eval(ts(end));
[~,~,~,dxp] = p.collisionDynamics(1,0,xend,0);
Ad = dxp(:,3:end-1);

% S_t_minus = inv(Ad'*inv(S_t_plus)*Ad + Q);
S_t_minus = Ad'*S_t_plus*Ad;


% Set Qf = to S_t_minus
Qf = S_t_minus;

% Check for convergence (FILL ME IN)
if all(all(abs(Qf - QfV) < 1e-4))
    converged = true;
end



end

Qf_converged = Qf

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Setup tvlqr controller and simulate from x0

% Extend nominal limit cycle a little bit (see footnote at the bottom for
% an explanation of this)
tExtended = 1.0;
xtrajExtended = p.modes{1}.simulate([0 tExtended],x0);
utrajExtended = PPTrajectory(spline(linspace(0,tExtended,100),zeros(1,100)));
xtrajExtended = xtrajExtended.setOutputFrame(p.modes{1}.getStateFrame);
utrajExtended = utrajExtended.setOutputFrame(p.modes{1}.getInputFrame);
```

```
[tv,V] = tvlqr(p.modes{1},xtrajExtended,utrajExtended,Q,R,Qf,options);

% Set frames of tvlqr controller
tv = tv.inOutputFrame(p.getInputFrame);
tv = tv.inInputFrame(p.getOutputFrame);

pmodel = SimulinkModel(p.getModel());

tv = tv.setInputFrame(pmodel.getOutputFrame);
tv = tv.setOutputFrame(pmodel.getInputFrame);

% Closed loop system (i.e., feedback system with TVLQR controller)
sysClosedLoop = feedback(pmodel,tv);

% Visualizer
v = CompassGaitVisualizer(p);

%% Simulate from x0 (syntax is useful for part (b)) %%%%%%%%%%%%%%%%%%%%%%
xtrajSim = sysClosedLoop.simulate([0 1.0], [1;xtraj.eval(0)]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Play trajectory back
xtrajSim = xtrajSim.setOutputFrame(p.getOutputFrame);
v.playback(xtrajSim);

%% Footnote (feel free to ignore) %%
% Since the continuous portion of our nominal trajectory is only defined for a finite amount of
% [0,ts(end)], we need to deal with cases where the compass gait doesn't
% make contact with the ground before ts(end). Our solution here is to
% extend the nominal trajectory (xtrajExtended) a little beyond ts(end) (by just simulating
% the passive system forwards for longer). This is a reasonable thing to do,
% but still somewhat of a hack. However, this is an important
% implementation issue and worth thinking about in practice.
%%
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Controlling the Compass Gait (Part II)

0.0/7.0 points (ungraded)

(b) Next, implement a function similar to the function strideFunction in the previous problem in order to simulate the closed-loop system for one step. The code we have provided you in stabilizeCompassGait.m shows you how to simulate the closed loop system ("sysClosedLoop in the code"). Use this to compute the Floquet multipliers for the closed-loop system just as you did in the previous problem for the passive system. Type in your answer below as a 4×1 vector of complex numbers. Don't worry about ordering the multipliers in any particular order. **Our tolerance for this answer is 0.05 (in terms of the magnitude of each element in the vector).** (Compare the largest Floquet multiplier with the largest multiplier you obtained for the passive system in part (d) of the previous problem. Is it smaller or bigger in magnitude? Is this what you expected?)

```
1 f = ;
2
```

Unanswered

```
% We can compute the monodromy matrix and Floquet multipliers just like the last problem.
% The monodromy matrix is:
A = [0.1281    1.7540   -0.0694    0.5891
    -0.1281   -1.7540    0.0694   -0.5891
    -1.5076   -2.9420    0.2520   -1.2411
    -0.2308    2.6514   -0.0817    0.9266];

% The Floquet multipliers are:
f = [-0.4729 + 0.0000i
      0.0128 + 0.1589i
      0.0128 - 0.1589i
     -0.0000 + 0.0000i];
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

6.832 Midterm

Name: SOLUTIONS

October 23, 2014

Please do not open the test packet until you are asked to do so.

- You will be given 85 minutes to complete the exam.
- Please write your name on this page, and on any additional pages that are in danger of getting separated.
- We have left workspace in this booklet. Scrap paper is available from the staff. Any scrap paper should be handed in with your exam.
- YOU MUST WRITE ALL OF YOUR ANSWERS IN THIS BOOKLET (not the scrap paper).
- The test is open notes.
- The test is out of 35 points.

Good luck!

Problem	Possible	Your Score
Problem 1	5	
Problem 2	10	
Problem 3	10	
Problem 4	10	
Total	35	

Problem 1 (5 pts) *The Linear Quadratic Regulator*

Suppose you are given a stabilizable linear system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$

and the cost-to-go function

$$J(\mathbf{x}_0) = \int_0^\infty \left[\mathbf{x}(t)^T\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^T(t)\mathbf{R}\mathbf{u}(t)\right] dt, \quad \mathbf{x}(0) = \mathbf{x}_0,$$

with $\mathbf{Q}^T = \mathbf{Q} \succ 0$, $\mathbf{R}^T = \mathbf{R} \succ 0$, and $eig(\mathbf{A}) \neq 0$. I can plot the level-set of the optimal cost-to-go function $J^*(x) = 1$; for this problem formulation the optimal cost-to-go is a positive-definite quadratic function, $J^* = \mathbf{x}^T\mathbf{S}\mathbf{x}$, $\mathbf{S} = \mathbf{S}^T \succ 0$ and the level-set is an ellipse. The optimal control is given by $\mathbf{u}^* = -\mathbf{K}\mathbf{x}$.

a) *If I were to double the value of $\mathbf{R}$ (so $\mathbf{R}_{new} = 2\mathbf{R}$), what happens to the 1 level-set of the cost-to-go? (Circle one)*

   (i) It gets bigger (the volume of the ellipse increases)

   (ii) It gets smaller (the volume of the ellipse decreases)

   (iii) The size does not change (the volume of the ellipse remains constant)

   Give a mathematical justification for your answer:

   Solution: *The answer is (ii) - it gets smaller. Intuitively, by increasing the instantaneous cost in this way, the cost-to-go can only increase. To see this formally, write the cost function as*

$$J_{new}(\mathbf{x}_0) = \int_0^\infty \left[\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right] dt + \int_0^\infty \mathbf{u}^T\mathbf{R}\mathbf{u}dt$$

   To be very clear, we let us write $J(\mathbf{x}_0, \mathbf{K})$ for the cost of executing with the controller $\mathbf{u} = -\mathbf{K}\mathbf{x}$. We would like to compare the cost-to-go $J_{new}(\mathbf{x}_0, \mathbf{K}^*_{new})$ to the original optimal cost-to-go $J(\mathbf{x}_0, \mathbf{K}^*)$.

   First observe that the optimal control gains for the new problem, $\mathbf{K}^*_{new}$, cannot produce a lower cost on the original problem, otherwise they would have outperformed the original optimal controller, so

$$J(\mathbf{x}_0, \mathbf{K}^*_{new}) \geq J(\mathbf{x}_0, \mathbf{K}^*).$$

   Furthermore, since $\mathbf{R}$ is strictly positive definite, we have

$$J_{new}(\mathbf{x}_0, \mathbf{K}^*_{new}) \geq J(\mathbf{x}_0, \mathbf{K}^*_{new}).$$

   Note that by giving $\mathbf{Q}$ strictly positive definite and $\mathbf{A}$ non-zero, we assure that the optimal control gains $\mathbf{K}^*_{new}$ are non-zero, which gives $J_{new}(\mathbf{x}_0, \mathbf{K}^*_{new}) > J(\mathbf{x}_0, \mathbf{K}^*_{new})$ (except for equality at $\mathbf{x}_0 = 0$).

   Since the cost-to-go is quadratic and strictly greater than the original cost-to-go, the 1-level set is strictly smaller.

b) Now suppose that you have doubled both $\mathbf{Q}$ *and* $\mathbf{R}$*, so*

$$\mathbf{Q}_{new} = 2\mathbf{Q}, \quad \mathbf{R}_{new} = 2\mathbf{R}.$$

Which of the following statements about the resulting optimal cost-to-go,

$$J^* = \mathbf{x}^T \mathbf{S}_{new} \mathbf{x},$$

are true (circle all that apply).

(i) $\mathbf{S}_{new} = \frac{1}{4}\mathbf{S}$

(ii) $\mathbf{S}_{new} = \frac{1}{2}\mathbf{S}$

(iii) $\mathbf{S}_{new} = \mathbf{S}$

(iv) $\mathbf{S}_{new} = 2\mathbf{S}$

(v) $\mathbf{S}_{new} = 4\mathbf{S}$

Give a mathematical justification for your answer:

Solution: *The answer is (iv) -* $\mathbf{S_{new} = 2S}$*. Intuitively, we would expect that if we simply double the instantaneous cost then we would also simply double the cost-to-go. Mathematically, we can see that if* $\mathbf{S}$ *is a solution to the algebraic Riccati equation,*

$$0 = Q - SBR^{-1}B^T S + SA + A^T S,$$

then $\mathbf{S}_{new} = 2\mathbf{S}$ *must be a solution to*

$$0 = 2Q - S_{new}B(2R)^{-1}B^T S_{new} + S_{new}A + A^T S_{new}.$$

c) Continuing with the formulation in part (b), which of the following statements about the resulting optimal controller,

$$\mathbf{u}^*_{new} = -\mathbf{K}_{new}\mathbf{x},$$

are true (circle all that apply):

(i) $\mathbf{K}_{new} > \mathbf{K}$

(ii) $\mathbf{K}_{new} = \mathbf{K}$

(iii) $\mathbf{K}_{new} < \mathbf{K}$

where the inequality is taken element-wise. Remember the $-\mathbf{K}$ *in the description above.*

Give a mathematical justification for your answer:

Solution: *The answer is (ii) -* $\mathbf{K}_{new} = \mathbf{K}$*. Following the solution to part (b), we know that*

$$\mathbf{u}^*_{new} = -(2\mathbf{R})^{-1}\mathbf{B}^T\mathbf{S}_{new}\mathbf{x} = -\mathbf{K}\mathbf{x}.$$

This is as we would expect - doubling the cost doubles the cost-to-go, but does not actually change the optimal policy.

Problem 2 (10 pts) *Lyapunov analysis*

a) Suppose you have a system $\dot{\mathbf{x}} = f(\mathbf{x})$ with $f(0) = 0$ and a positive-definite scalar function $V(\mathbf{x})$ where you have successfully verified that

$$\dot{V}(0) = 0$$
$$\dot{V}(\mathbf{x}) < 0, \quad \forall \mathbf{x} \text{ with } 0 < \sum_i |x_i| \leq 1.$$

Describe the set of initial conditions for which you can guarantee that the system will arrive at the origin as $t \to \infty$. Explain your answer.

Solution: *The conditions above guarantee that V does not increase while the system is inside the region described (which you may have recognized as the unit ball for the l_1 norm). But we can only guarantee convergence for trajectories which* never leave *this region. The best we can guarantee is the sub-level set, $V \leq \rho$ where ρ is the* smallest *value of V on the boundary described by $\sum_i |x_i| = 1$.*

b) Consider an uncertain nonlinear system of the form

$$\dot{\mathbf{x}} = f_1(x) + \alpha f_2(\mathbf{x}), \quad \alpha = \{0.8, 1.1\}.$$

In words, the uncertain gain α is known to take one of exactly two values – either 0.8 or 1.1, but we do not know apriori which one.

(i) Suppose that you know that the origin, $\mathbf{x} = 0$, is a fixed point for the system $\dot{\mathbf{x}} = f_1(\mathbf{x}) + f_2(\mathbf{x})$. Is the origin guaranteed to be a fixed point for the uncertain system? Circle yes or no.

YES *or* NO

Explain your answer:

Solution: *The answer is NO.*

$$\dot{\mathbf{x}} = 0 \Rightarrow f_1(\mathbf{x}) = -f_2(\mathbf{x}),$$

but does not say anything about $\alpha f_2(\mathbf{x})$.

(ii) Suppose that you are given a radially-unbounded, positive-definite Lyapunov function, $V(\mathbf{x})$, which satisfies the conditions

$$\forall \mathbf{x} \neq 0, \dot{V}(\mathbf{x}, 0.8) < 0, \quad \dot{V}(0, 0.8) = 0,$$
$$\forall \mathbf{x} \neq 0, \dot{V}(\mathbf{x}, 1.1) < 0, \quad \dot{V}(0, 1.1) = 0,$$

where I've used the notation $\dot{V}(\mathbf{x}, \alpha) = \frac{\partial V}{\partial \mathbf{x}} \left[f_1(\mathbf{x}) + \alpha f_2(\mathbf{x}) \right]$.
Which of the following can we conclude about the system

$$\dot{\mathbf{x}} = f_1(\mathbf{x}) + f_2(\mathbf{x})$$

i. The origin is globally stable in the sense of Lyapunov (i.s.L.).
ii. The origin is globally asymptotically stable.
iii. The origin is globally exponentially stable.
iv. None of the above.

Circle all that are true. Provide a mathematical justification for your answer:

Solution: *The first two are true (i and ii). This can be demonstrated by realizing that the original system can be written as an affine combination of the two systems, and therefore we have*

$$\dot{V}(\mathbf{x}, 1) = \frac{1}{3}\dot{V}(\mathbf{x}, .8) + \frac{2}{3}\dot{V}(\mathbf{x}, 1.1) < 0,$$

for $\mathbf{x} \neq 0$ and $\dot{V}(0, 1) = 0$. Since V is radially-unbounded this satifies the conditions to demonstrate global asymptotic stability of the origin, and also stability i.s.L.
Note that this may be surprising given the result in part (a), but in fact knowing that V is strictly decreasing for both systems away from $\mathbf{x} = 0$ implies that $\mathbf{x} = 0$ is the fixed point for the original system as well.

(iii) Suppose that you are given a radially-unbounded, positive-definite Lyapunov function candidate, $V(\mathbf{x})$, which you know fails to satisfy the Lyapunov conditions globally, but you would like use sums-of-squares (SOS) optimization to verify the conditions:

$$\forall \mathbf{x} \in \{\mathbf{x} : V(\mathbf{x}) \leq 1\}, \dot{V}(\mathbf{x}, 0.8) \leq 0$$
$$\forall \mathbf{x} \in \{\mathbf{x} : V(\mathbf{x}) \leq 1\}, \dot{V}(\mathbf{x}, 1.1) \leq 0.$$

Write down a sums-of-squares program by listing the decision variables and all *of the required sums-of-squares constraints in the boxes below.*

find

$\lambda_1(\mathbf{x}), \lambda_2(\mathbf{x})$

parameterized by $\lambda_1(\mathbf{x}) = a^T m_1(\mathbf{x})$ and $\lambda_2(\mathbf{x}) = b^T m_2(\mathbf{x})$, where a and b are vectors of decision variables and $m_1(\mathbf{x})$ and $m_2(\mathbf{x})$ are monomial vectors.

(list decision variables)

subject to

$$-\dot{V}(\mathbf{x}, 0.8) + \lambda_1(\mathbf{x})(V(\mathbf{x}) - 1),$$
$$\lambda_1(\mathbf{x}),$$
$$-\dot{V}(\mathbf{x}, 1.1) + \lambda_2(\mathbf{x})(V(\mathbf{x}) - 1),$$
$$\lambda_2(\mathbf{x})$$

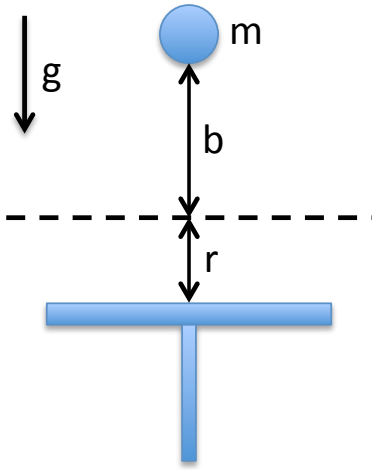
is SOS

Note that it is also acceptable to use the same multipliers for both systems, this would potentially be more conservative but would also use less decision variables so could allow you to search over more monomial vector coefficients.

This page intentionally left blank for workspace

Problem 3 (10 pts) *Trajectory optimization*

Imagine the simple model of a juggling robot illustrated below, which consists of a paddle that can move vertically (with configuration described by the position r) and a ball (modeled as a point with mass m at vertical height b).



Our input is direct control of the velocity of the paddle, resulting in a hybrid state space model with

$$\mathbf{x} = \begin{bmatrix} r \\ b \\ \dot{b} \end{bmatrix}, \quad \dot{\mathbf{x}} = f(\mathbf{x}, u) = \begin{bmatrix} u \\ \dot{b} \\ -g \end{bmatrix},$$

and a simple elastic collision model,

$$\dot{b}^+ = \dot{r} - .9(\dot{b}^- + \dot{r}).$$

We assume the paddle is sufficiently massive to be unaffected by the collision with the ball.

Let us formulate a direct transcription trajectory optimization problem with 11 *knot points to find a periodic solution for this system with exactly one collision per period. We will use the decision variables*

$$\mathbf{x}_0, ...\mathbf{x}_{10}, \quad u_0, ..., u_9, \quad h.$$

and add the dynamic constraints

$$\mathbf{x}_{n+1} = \mathbf{x}_n + f(\mathbf{x}_n, u_n)h,$$

where $h > 0.01$ *is the timestep between knot points.*

a) *Let us enforce that the paddle and the ball only come into contact at height 0. Write down all of the required constraints that you must add to the program relating to the hybrid guard and reset (aka collision). Your answers should be written in terms of differentiable functions of the decision variables named above.*

Solution: *We can accomplish this by, e.g., requiring that the ball always stays above vertical and the paddle always stays below the ball (except at the endpoints):*

$$b_0 = b_{10} = 0$$
$$r_0 = r_{10} = 0$$
$$r_1 < b_1, r_2 < b_2, ..., r_9 < b_9$$

and we add the periodicity/reset constraint

$$\dot{b}_0 = u_0 - .9(\dot{b}_{10} + u_0).$$

Note that $b_1, ..., b_9 > 0$ will be enforced automatically given the dynamic constraints (but it does no harm to include it).

b) *To further constrain the solution, let us require that the apex of the ball occurs at a height of 1. Write any additional constraints required to enforce this. Your answers should be written in terms of differentiable functions of the decision variables named above.*

Solution: *While there are many creative ways to enforce this constraint, the simplest by far is to realize that the solution ball dynamics during the aerial phase will always be symmetric in time (it is the ballistic trajectory of a point mass). Therefore, it is sufficient to add the constraint*

$$b_5 = 1.$$

Note that $\dot{b}_5 = 0$ will be enforced automatically by the dynamic constraints (but it does no harm to include it).

c) Do you expect the problem you have formulated so far to have a unique solution? If not, explain why and provide a reasonable additive-cost objective function written in terms of the decision variables.

Solution: *The problem so far has done nothing to describe the desired trajectory for the paddle, except to constrain the velocity at time* 0 *(and therefore* 10*) and say that we'd like the paddle to always be below the ball. Therefore, we can express our preference for the motion of the paddle through the objective function. A simple integral cost penalizing paddle velocity should do the trick:*

$$\sum_{i=0}^{9} u_i^2.$$

I would expect the result to be a paddle trajectory the matches the required velocities at the endpoints, and then evenly distributes over the remaining trajectory the velocity required to reset the paddle for the next hit.

This page intentionally left blank for workspace

Problem 4 (10 pts) *Nonlinear Dynamics and the Hamilton-Jacobi-Bellman Equations*

a) Suppose you are given the system

$$\dot{x} = x + u,$$

where x and u are scalars, and the cost function,

$$J(x_0) = \int_0^\infty g(x(t), u(t))dt, \quad g(x, u) = x^2 + u^2, \quad x(0) = x_0,$$

which we would like to minimize. Suppose you are also given the candidate cost-to-go function

$$\hat{J}(x) = x^4 - x^2 + \frac{1}{4}.$$

Use the optimality conditions to derive the control law associated with this cost-to-go:

$$\hat{u} = \underset{u}{\operatorname{argmin}} \left[g(x, u) + \frac{\partial \hat{J}}{\partial x} f(x, u) \right].$$

Show your work.

Solution: *Because the argmin is taken over a positive quadratic form in u, we can do the minimization by finding the u for which the gradient of the right-hand side (RHS) is zero:*

$$\hat{u} = \underset{u}{\operatorname{argmin}} \left[x^2 + u^2 + (4x^3 - 2x)(x + u) \right]$$

$$\frac{\partial RHS}{\partial u} = 2u + (4x^3 - 2x) = 0$$

$$\hat{u} = -2x^3 + x$$

b) Is the controller derived above optimal? Circle one of the following:

YES *or* NO *or* INCONCLUSIVE

Solution: *This was a tricky question. The most important point by far was to demonstrate that you understand the fact that the HJB condition is a sufficient condition, but not a necessary one. Substituting $\hat{J}$ and the $\hat{u}$ derived in part (a) into the HJB you will see that it does not match with equality. This would lead one to circle "Inconclusive". Circling "No" using only this justification is absolutely incorrect.*

But it turns out that we do know more for this problem. The dynamics are linear and the objective is a positive quadratic – so the problem formulation is LQR! Since LQR results in a linear optimal feedback and quadratic cost-to-go, we can definitively conclude that the correct solution is "NO"*, the controller is not optimal.*

c) Using the system and controller from part (a), what are the fixed points of the closed-loop system. For each fixed point, say if it is locally unstable, locally stable i.s.L, locally asymptotically stable, and/or locally exponentially stable.

Solution: *Using this controller, the closed-loop dynamics are*

$$\dot{x} = 2x - 2x^3.$$

This is just -2 times the cubic example used in the lecture notes. The fixed points are at $x = -1$,0, and 1. Sketching the one-dimesional flow, we can verify graphically that $x = -1$ and $x = 1$ are locally stable fixed points, and that $x = 0$ is locally unstable. Both stable fixed points are stable i.s.L, asymptotically, and exponentially, which can be demonstrated graphically by drawing a line between the $\dot{x}$-curve and the x-axis to represent an exponentially stable linear system which this system converges faster than.

Alternatively, we can verify the stability by evaluating the derivative at the fixed points, and noting that locally, the stable fixed points look like

$$\dot{\bar{x}} \approx (2 - 6(1^2))\bar{x} = -4\bar{x}.$$

d) Now consider a similar problem, but we will remove the input term from the cost function and add input limits to the system, so that we have

$$\dot{x} = x + u, \quad |u| \leq 1,$$

and the cost function,

$$J(x_0) = \int_0^\infty g(x(t), u(t))dt, \quad g(x, u) = x^2, \quad x(0) = x_0,$$

which we would like to minimize. Using the same candidate cost-to-go function

$$\hat{J}(x) = x^4 - x^2 + \frac{1}{4},$$

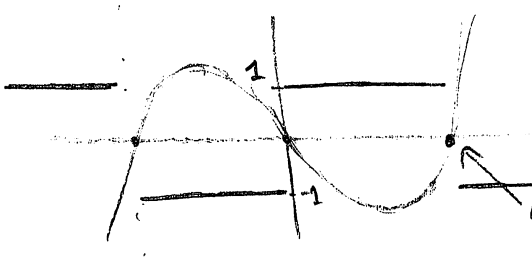
use the optimality conditions to derive the control law associated with the cost to go for this problem. Show your work.

Solution on next page

e) Using the system and controller from part (d), what are the fixed points of the closed-loop system. For each fixed point, say if it is locally unstable, locally stable i.s.L, locally asymptotically stable, and/or locally exponentially stable.

$$\hat{u} = \underset{u}{\operatorname{argmin}} \left[x^2 + (4x^3 - 2x)(x - u) \right].$$

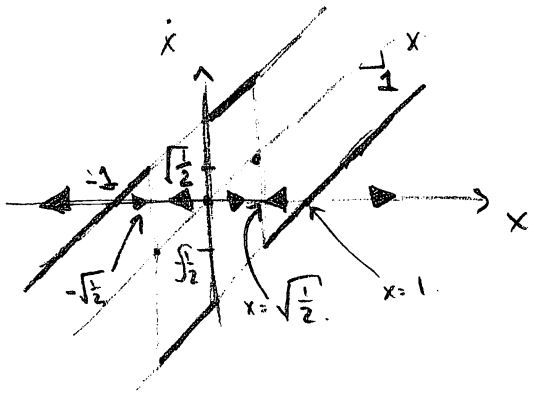
$$\hat{u} = -\operatorname{sgn}(4x^3 - 2x) = \begin{cases} -1 & 4x^3 - 2x > 0 \\ 0 & 4x^3 - 2x = 0 \\ 1 & 4x^3 - 2x < 0 \end{cases}$$



$4x^3 - 2x = 0 \Rightarrow x(2x^2 - 1) = 0$

$x = \pm\sqrt{\frac{1}{2}}, 0.$

$$\dot{x} = \begin{cases} x - 1 & \text{for } 4x^3 - 2x > 0 \\ x & = 0 \\ x + 1 & > 0 \end{cases}$$



The closed-loop system has unstable fixed points at $x = 0$, $x = -1$, and $x = 1$.

This page intentionally left blank for workspace

ZMP

Zero Moment Point

0.0/5.0 points (graded)

Consider the ZMP equation, which describes the center of mass dynamics of a walking robot.

$$x_{cm} - x_{zmp} = \frac{z_{cm}}{g}\ddot{x}_{cm}$$

where x_{cm} is the center of mass position, z_{cm} is the center of mass height, and x_{zmp} is the ZMP position.

To achieve this linear model, which assumptions/simplifications have been made? Select all that apply.

- [ ] The robot legs are massless
- [ ] $\dot{z}_{cm}$ is held constant. ✔
- [ ] The robot dynamics are linear
- [ ] Angular momentum is held constant. ✔

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

ZMP Planning: Part A

0.0/8.0 points (graded)

For this problem, we will use a simple ZMP plan to construct a nominal trajectory for the center of mass that tracks the ZMP plan. Construct a linear system where the state $\mathbf{x} = \begin{bmatrix} x_{cm} \\ \dot{x}_{cm} \end{bmatrix}$ is the position of the center of mass $x_{cm} \in \mathbb{R}^2$ (for 3-D walking) and the velocity $\dot{x}_{cm}$. Take the output of to be the position of the ZMP, $x_{zmp} \in \mathbb{R}^2$. Let the input $u = \ddot{x}_{cm}$. Write your system in the form:

$$\dot{\mathbf{x}} = A\mathbf{x} + Bu, \quad x_{zmp} = C\mathbf{x} + Du.$$

```
g = 9.81;   %gravity
z_cm = 1.1; %height of the center of mass
A =
B =
C =
D =

```

Unanswered

```
g = 9.81;   %gravity
z_cm = 1.1; %height of the center of mass

A = [zeros(2) eye(2); zeros(2,4)];
B = [zeros(2);eye(2)];
C = [eye(2) zeros(2)];
D = -eye(2)*z_cm/g;
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

ZMP Planning: Part B

0.0/8.0 points (graded)

With the system above, construct a linear quadratic optimal control problem to stablize the ZMP to the origin. Write the cost as $\int x_{zmp}^T Q_{zmp} x_{zmp}$

Convert this problem into standard LQR format, and call MATLAB's built in **lqr** function. See **help lqr** for an explanation of the term N. For $Q_{zmp} = I^{2\times 2}$, find the optimal cost-to-go S.

Rewrite this tracking problem in terms of the state and input, in the form

$$\int_0^\infty \mathbf{x}^T Q \mathbf{x} + q^T \mathbf{x} + u^T R u + r^T u + 2\mathbf{x}^T N u dt$$

where we have discarded any constant terms in the cost. Note the "2", this is to fit your solution into MATLAB's **lqr** format.

```
g = 9.81;   %gravity
z_cm = 1.1; %height of the center of mass
A =
B =
Q =
R =
N =

[K,S] = lqr(A,B,Q,R,N);

```

Unanswered

```
g = 9.81;   %gravity
z_cm = 1.1; %height of the center of mass

% an example of Q_zmp
Q_zmp = eye(2);

% A, B, C, and D from last question:
A = [zeros(2) eye(2); zeros(2,4)];
B = [zeros(2);eye(2)];
C = [eye(2) zeros(2)];
D = -eye(2)*z_cm/g;

Q = C'*Q_zmp*C;
R = D'*Q_zmp*D;
N = C'*Q_zmp*D;

[K,S] = lqr(A,B,Q,R,N);
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

ZMP Planning: Part C

0.0/9.0 points (graded)

Repeat Part B, but instead track a desired reference trajectory $x_{zmp}^{d}(t)$, where the cost is $\int (x_{zmp} - x_{zmp}^{d})^{T} Q_{zmp} (x_{zmp} - x_{zmp}^{d}) dt$

Rewrite this tracking problem in terms of the state and input, in the form

$$\mathbf{x}(T)^{T} Qf\mathbf{x}(T) + q_{f}^{T}\mathbf{x}(T) + \int_{0}^{T} \mathbf{x}^{T} Q(t)\mathbf{x} + q(t)^{T}\mathbf{x} + u^{T} R(t) u + r(t)^{T} u + 2\mathbf{x}^{T} N(t) u dt$$

where we have discarded any constant terms in the cost.

Download the stub code here, and implement the ZMP planner. For the final cost of the time-varying LQR problem, use the infinite horizon cost from Part B, but centered around the stationary end-point of the walking trajectory $(x_{cm} - x_{final})^{T} S (x_{cm} - x_{final})$.

A correct LQR controller should track the ZMP trajectory closely, and illustrate a smooth center of mass trajectory. For your working controller, what is $\ddot{y}_{cm}$ for $t = 1$ in the simulation? This should be the second element of u printed by the stub code.

Answer: .0376

Explanation

The time varying LQR costs are given by:

```
% The time varying costs for Q, R, N as in Part B.
% Now, however, the desired zmp position is zmp_traj, not a constant value (0),
% so we have linear terms as a result.
Q{1} = C'*Q_zmp*C;
Q{2} = -2*C'*Q_zmp*zmp_traj;
Q{3} = 0;


R{1} = D'*Q_zmp*D;
R{2} = -2*D'*Q_zmp*zmp_traj;
R{3} = 0;

options.Ny = C'*Q_zmp*D;

% Solve LQR at the final state to get infinite horizon cost
[~,S] = lqr(A,B,Q{1},R{1},options.Ny)
Qf{1} = S;
Qf{2} = -2*S*[zmp_traj.eval(T);0;0];
Qf{3} = 0;
```

Experiment by changing the walking speed and the amount of time at the beginning and end of the trajectory where the ZMP is motionless (t_beginning and t_end) in the code.

Submit You have used 0 of 3 attempts

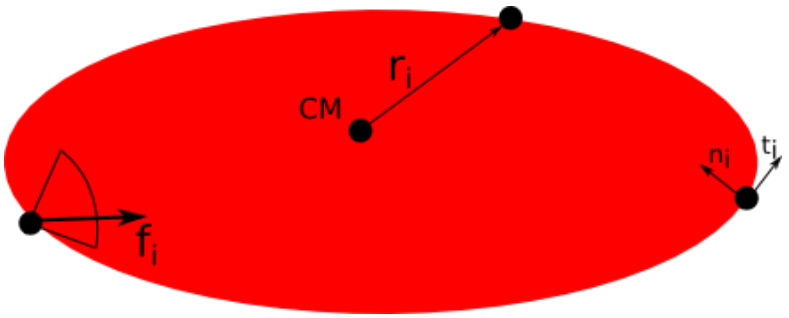
Answers are displayed within the problem

Force Closure

Force Closure

0.0/20.0 points (graded)

Force closure is an important concept in manipulation. Given a configuration with a robot grasping an object at some set of contact points, we say there is *force closure* if the robot can exert an arbitrary force and torque on the object by applying forces at the contact points. Note that for this problem, we will not consider the kinematics or dynamics of the robot itself--merely the quality of the grasp, as determined by the contact points.



For simplicity, we will consider a planar model, where the contact forces $f_i \in \mathbb{R}^2$. Write the forces in the frame of the contact, so the set of feasible forces within the friction cone is expressed as $f_i = \begin{bmatrix} f_{i,x} \\ f_{i,z} \end{bmatrix}$ with $|f_{i,x}| \leq \mu f_{i,z}$ and $f_{i,z} \geq 0$, observing that these can be rewritten as linear constraints.

Then, the net wrench w (the combined forces and torques) on the object is

$$w = \begin{bmatrix} \sum_i (f_{i,x} t_i + f_{i,z} n_i) \\ \sum_i r_i \times (f_{i,x} t_i + f_{i,z} n_i) \end{bmatrix}$$

It is fairly easy to see that we can write $w = Gf$, for some matrix G (depending on r_i, n_i, t_i) where f is the stacked vector of forces.

It can also be shown that the question of force closure can be reduced to this problem:

- Check that G is full rank,

- find f such that $Gf = 0$
- and f is in the *interior* of the set of allowable forces. Otherwise stated, $f_{i,z} > 0, \quad |f_{i,x}| < \mu f_{i,z}$ for all i.

The rank condition can easily be checked, but the other two points are slightly more difficult. Assuming that G is full rank, we would like to write a Linear Program (LP) that checks, for some r_i's, whether or not a grasp has force closure. Recall that an LP is an optimization of the form:

$$\begin{aligned} \min_z & c^T z \\ \text{s.t.} & Az = b \\ & Cz \leq d \end{aligned}$$

For some matrices A and C and vectors c, b, d.

The problem: write an LP such that $c^T z = 0$ if and only if there is **no** force closure, and $c^T z < 0$ when there is force closure.

HINT: The strict inequalities (> and <) in the description above can be tricky to incorporate into this format. Introduce a slack variable $\gamma \leq 0$ and let $z = \begin{bmatrix} \gamma \\ f \end{bmatrix}$ and c such that $c^T z = \gamma$. Incorporate γ into the constraints above such that $\gamma < 0$ implies that $f_{i,z} > 0$ and $|f_{i,x}| < \mu f_{i,z}$ for all i.

Make sure that your LP is bounded (that there is no feasible z such that $c^T z = \infty$)! You can also use MATLAB's **linprog** function to test your LP on numeric values, but careful for their different naming conventions.

```
% PROBLEM SETUP, DO NOT EDIT
% Construct vectors r1 and r2
r1=sym('r1',[2 1]);
sym(r1,'real');

r2=sym('r2',[2 1]);
sym(r2,'real');

n1=sym('n1',[2 1]);
sym(n1,'real');

t1=sym('t1',[2 1]);
sym(t1,'real');

n2=sym('n2',[2 1]);
sym(n2,'real');
```

Unanswered

```
% decision variables [gamma;f1;f2]
c = [1; zeros(4,1)];

% sum forces and torque in inertial frame
G = [0 t1(1) n1(1) t2(1) n2(1);
     0 t1(2) n1(2) t2(2) n2(2);
     0, t1(2)*r1(1) - t1(1)*r1(2), n1(2)*r1(1) - n1(1)*r1(2), t2(2)*r2(1) - t2(1)*r2(2),
A = G;
b = zeros(3,1);

% create C and d for friction cone
C = zeros(0,5);
C = [C;-1 0 -1 0 0];  %f_1z + gamma >= 0
C = [C;-1 1 -mu 0 0]; %mu f_1z + gamma >= f_1x
C = [C;-1 -1 -mu 0 0]; %f_1x + gamma >= -mu f_1z


% same for f2
C = [C;-1 0 0 0 -1]; %f_2z + gamma >= 0
C = [C;-1 0 0 1 -mu]; %mu f_1z + gamma >= f_1x
C = [C;-1 0 0 -1 -mu]; %f_1x + gamma >= -mu f_1z

% bound gamma
C = [C;-1 0 0 0 0];  % gamma >= -1

d = [zeros(6,1);1];
```

Run Code

As an aside, if we wanted to implement this same check in 3D, the friction cone constraint is no longer linear! Standard approaches to this are to form a polyhedral approximation of the friction cone, or to perform the check as a Second Order Cone Program (SOCP), which is another type of convex optimization.

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Feasible Motion Planning

Feasible Motion Planning

0.0/20.0 points (graded)

This question will test your understanding of A* planning and Rapidly Exploring Randomized Trees (RRTs) with a series of short questions. **Unless mentioned otherwise, all the planning problems considered here are in continuous state and action spaces.**

(a) Suppose we have a bounded environment where there is **no** collision free path from the start to the goal. Which of the following is true?

- [ ] The RRT algorithm will terminate in finite time.
- [ ] The A* algorithm (with a fixed grid resolution) will terminate in finite time. ✔

Explanation

The first choice is incorrect since the RRT algorithm will keep going if it doesn't find a feasible path. The only guarantee we can make for the RRT is that it is probabilistically complete, i.e., that if there is a feasible path, the probability that one is found approaches 1 as time approaches infinity.

The second choice is correct since with a fixed resolution the planning problem is finite and thus the A* will terminate in finite time.

(b) Which of the following statements about RRTs are true in general? Select all that apply. (Here, you may assume that the obstacle set and free space are "nice". In particular, we assume that the obstacle set is a closed set in the topological sense).

- [ ] The algorithm is complete (i.e., the algorithm will always terminate in finite time. Further, it will return a feasible path if and only if one exists).

- [x] The algorithm is probabilistically complete (i.e., if there is a feasible path, the probability that one is found approaches 1 as time approaches infinity). ✔

- [ ] Assume that a feasible motion plan exists. Then there exists a $T \in \mathbb{R}$ such that the probability that a feasible path is found within time T is 1.

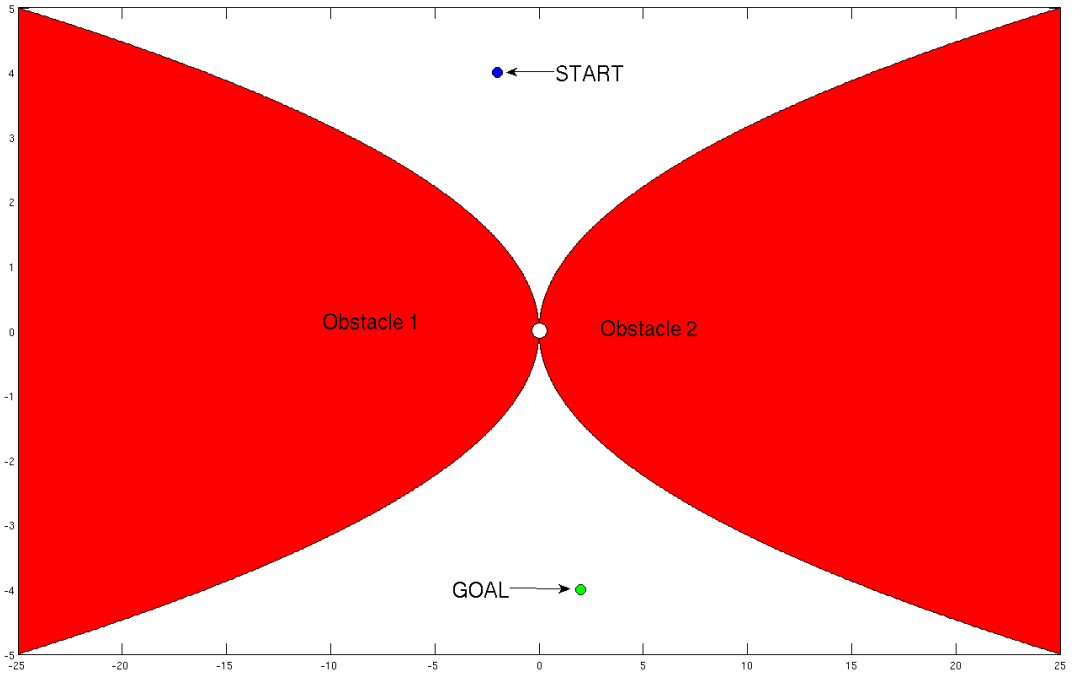
- [ ] As time approaches infinity, the probability that an **optimal** path is found approaches 1 (here "optimal" means shortest path).

- [ ] Assume that a feasible motion plan exists. As time approaches infinity, the probability that a feasible path is found is bounded above by a constant $c < 1$.

Explanation

All choices except the second one are incorrect. The only guarantee we can make for the RRT is that if there is a feasible path, the probability that one is found approaches 1 as time approaches infinity.

(c) Suppose we have a planning problem in $\mathbb{R}^2$ where there are two obstacles. The first obstacle is the set $\{(x, y) \mid (x, y) \neq (0, 0), x \leq -y^2\}$ and the second obstacle is the set $\{(x, y) \mid (x, y) \neq (0, 0), x \geq y^2\}$. These obstacles are shown in the figure below, as are the start and goal positions. Notice that this motion planning problem is feasible (i.e., there exists a collision free path from start to goal).



Suppose we use the RRT algorithm to solve this planning problem. As time goes to infinity, which of the following is true?

- The probability that a feasible path is found approaches 0. ✔
- The probability that a feasible path is found approaches some number in the open interval $(0, 1)$.
- The probability that a feasible path is found approaches 1.

Explanation

The first choice is correct. In other words, the RRT algorithm will fail for this problem. This is because in order to successfully find a feasible path, the algorithm must sample a point on the line $x = 0$ and then perform the "extend operation" vertically downwards (with no margin for error). The probability that this occurs is 0.

(d) In this question, we will consider the planning problem on a uniform two dimensional grid (here "uniform" means that the grid cells are squares). Suppose we are given a number of obstacles along with start and goal cells. The actions available are "up", "down", "left" and "right". Which of the following are *admissible heuristics* in general for the A* algorithm (recall that an admissible heuristic is one that lower bounds the true cost)?

- [ ] Euclidean distance in $\mathbb{R}^2$. ✔
- [ ] Euclidean distance squared.
- [ ] L_1 distance (also known as the "taxicab distance". The distance between points p and q in two dimensions is given by $L_1(p,q) = \sum_{i=1}^{2} |p_i - q_i|$.). ✔
- [ ] L_∞ distance (sometimes known as the "Chebyshev distance". The distance between points p and q in two dimensions is given by $L_\infty(p,q) = \max\{|p_1 - q_1|, |p_2 - q_2|\}$.). ✔
- [ ] The heuristic that is identically equal to 0. ✔
- [ ] The heuristic that is identically equal to 100.

Explanation

An admissible heuristic must underestimate the optimal distance. This is not the case for the square of the Euclidean distance (e.g., if the distance is very large). This is also not true for the heuristic that returns a non-zero constant (e.g., 100). All other choices are admissible heuristics.

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

RRT

RRT (Part A)

0.0/10.0 points (graded)

In this problem, we will implement a Rapidly Exploring Randomized Tree (RRT). For easy visualization, we will consider a two dimensional example. However, it will be relatively straightforward to extend your code to handle higher dimensional environments.

(a) The most computationally expensive portion of the algorithm is typically the collision checker. Write code that will take a set of points $\{x_i\}$ in two dimensions and determine if a given set of points $\{x_{0,i}\}$ **are in the convex hull** of the points $\{x_i\}$. The output of your code should be a row vector of boolean variables ("collisionFree" in the code below) whose i^{th} element is "true" if and only if $x_{0,i}$ is **not** in the convex hull.

Hint: The Matlab functions convhull and inpolygon may be useful to you here (but of course, you don't have to use them). Also, check your code by making plots a few times before submitting it.

```
%%% Problem Setup (DO NOT MODIFY) %%%
xi = randn(2,10); % Each column is a point (x,y)
x0 = randn(2,10);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Your code here %%%%%%%%%%%%%%%%%%

collisionFree = ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Unanswered

```
xi = randn(2,10); % Each column is a point (x,y)
x0 = randn(2,10);

% Get convex hull of points
K = convhull(xi(1,:),xi(2,:));
xobs = xi(1,K);
yobs = xi(2,K);

% Check to see if x0 is in polygon
in = inpolygon(x0(1,:),x0(2,:),xobs,yobs);

collisionFree = ~in;
```

Run Code

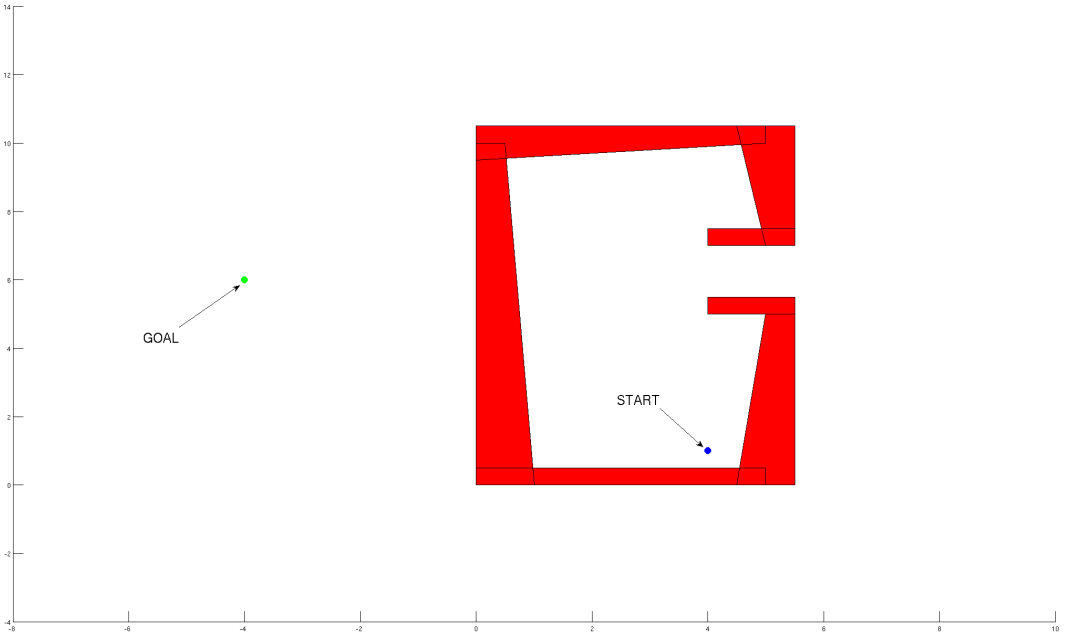
Submit You have used 0 of 3 attempts

Answers are displayed within the problem

RRT (Part B)

0.0/20.0 points (graded)

(b) Next, we will implement the entire RRT algorithm and test it out on a "bug trap" environment (shown in the figure below). This kind of environment is typically quite challenging for motion planning algorithms since it requires discovering and navigating through a narrow passage in order to get out of the "trap". However, hopefully you'll see that the RRT algorithm is able to do this with relative ease.



We have provided some stub code here. Fill in the portions that say "FILL ME IN". Run your code a few times to get intuition for how the RRT grows. Once you are satisfied that the code is running correctly, paste in the RRT your code found below (this is the variable "rrt_verts_grade" in the code). The first row should be the start state and the last row should be (close to) the goal state. You may find the "clipboard" Matlab function useful for copying data in variables.

```
rrt_verts_grade = [];

```

Unanswered

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for finding closest vertex in tree
function closest_vert = closestVertex(rrt_verts,xy)

% Find distances from points in rrt_verts to xy
diffs = rrt_verts - repmat(xy,1,size(rrt_verts,2));
dists = sqrt(diffs(1,:).^2 + diffs(2,:).^2);

% Find closest vertex
[~,minind] = min(dists);

closest_vert = rrt_verts(:,minind);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for collision checking
function collFree = isCollisionFree(Obs,xy)

collFree = zeros(length(Obs),1);
for k = 1:length(Obs)
    % Convex hull
    K = convhull(Obs{k}(1,:),Obs{k}(2,:));
    xobs = Obs{k}(1,K);
    yobs = Obs{k}(2,K);

    % Check if point is inside polygon
    in = inpolygon(xy(1),xy(2),xobs,yobs);

    if ~in
        collFree(k) = 1;
    end


end

collFree = all(collFree);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rrt_bugtrap.m

%% Code setup (Do not modify, but please read) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Obstacles. Each obstacle is a cell in Obs. An obstacle is
% represented as a convex hull of a number of points. These points are
```

```
% stored in the cells of Obs.
% First row is x, second is y (position of vertices)
w = 0.5;
Obs{1} = [0 0;5 0;5 w;0 w]';
Obs{2} = [0 0;2*w 0;w 10;0 10]';
Obs{3} = [0 10-w;5 10;5 10+w;0 10+w]';
Obs{4} = [5-w 0;5+w 0;5+w 5;5 5]';
Obs{5} = [5-w 10+w;5+w 10+w;5+w 7;5 7]';
Obs{6} = [4 5;5+w 5;5+w 5+w;4 5+w]';
Obs{7} = [4 7;5+w 7;5+w 7+w;4 7+w]';

% Bounds on world
world_bounds_x = [-8,10];
world_bounds_y = [-4,14];


% Draw obstacles
figure(1); clf; hold on;
axis([world_bounds_x world_bounds_y]);

for k = 1:length(Obs)
    patch(Obs{k}(1,:),Obs{k}(2,:),'r');
end

% Start and goal positions
xy_start = [4;1]; plot(xy_start(1),xy_start(2),'bo','MarkerFaceColor','b','MarkerSize',10);
xy_goal = [-4;6]; plot(xy_goal(1),xy_goal(2),'go','MarkerFaceColor','g','MarkerSize',10);

% Initialize RRT. The RRT will be represented as a 2 x N list of points. So
% each column represents a vertex of the tree.
rrt_verts = zeros(2,1000);
rrt_verts(:,1) = xy_start;
N = 1;
nearGoal = false; % This will be set to true if goal has been reached
minDistGoal = 0.25; % This is the convergence criterion. We will declare
                    % success when the tree reaches within 0.25 in distance
                    % from the goal. DO NOT MODIFY.

% Extension parameter
d = 0.5; % This controls how far the RRT extends in each step. DO NOT MODIFY.

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% RRT algorithm
while ~nearGoal
    % Sample point
    rnd = rand(1);
    % With probability 0.05, sample the goal. This promotes movement to the
```

```
% goal.
if rnd < 0.05
    xy = xy_goal;
else
    % Sample from space with probability 0.95
    xs = (world_bounds_x(2) - world_bounds_x(1))*rand(1) + world_bounds_x(1);
    ys = (world_bounds_y(2) - world_bounds_y(1))*rand(1) + world_bounds_y(1);
    xy = [xs;ys];
end

%% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check if sample is collision free
collFree = isCollisionFree(Obs,xy); % Write this function.
                                    % Your code from part (a) will be useful here.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% If it's not collision free, continue with loop
if ~collFree
    continue;
end

%% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If it is collision free, find closest point in tree
closest_vert = closestVertex(rrt_verts(:,1:N),xy); % Write this function

% Extend tree towards xy from closest_vert
new_vert = closest_vert + d*(xy - closest_vert)/norm(xy - closest_vert);

% Check if new_vert is collision free
collFree = isCollisionFree(Obs,new_vert); % Same function you wrote before.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



% If it is not collision free, continue with loop
 if ~collFree
    continue;
 end

% Plot extension (Comment the next 3 lines out if you want your code to
% run a bit quicker. The plotting is useful for debugging though.)
figure(1)
plot(new_vert(1),new_vert(2),'bo','MarkerFaceColor','b','MarkerSize',5);
line([closest_vert(1),new_vert(1)],[closest_vert(2),new_vert(2)]);


%% DO NOT MODIFY THIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If it is collision free, add it to tree
```

```
    N = N+1;
    if N > size(rrt_verts,2)
        rrt_verts = [rrt_verts zeros(size(rrt_verts))];
    end
    rrt_verts(:,N) = new_vert;

    % Check if we have reached goal
    if norm(xy_goal-new_vert) < minDistGoal
        break;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

% Plot vertices in RRT
plot(rrt_verts(1,:),rrt_verts(2,:),'bo','MarkerFaceColor','b','MarkerSize',5);

%% Submit rrt_verts_grade for grading %%%%
format long
rrt_verts_grade = rrt_verts(:,1:N)'
clipboard('copy',rrt_verts_grade);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

RRT with dynamic constraints

RRT with dynamic constraints (Part A)

0.0/5.0 points (ungraded)

In this problem, we will consider extensions of the RRT algorithm that allow us to handle dynamic constraints. In particular, we will use the swing-up problem for a torque-limited pendulum as our test-bed. The dynamics for the pendulum will be:

$$\ddot{\theta} = u - g\sin(\theta) - b\dot{\theta},$$

where $g = 9.81$, $b = 0.1$ and u **is bounded in the range [-5,5]**.

We have provided some stub code here. The basic structure of the RRT code is very similar. The only differences will be in the way we find the closest vertex in our existing tree (given a new sample) and the way we extend the tree. In particular, the extension operation must satisfy the dynamic constraints of the system. Please take a quick look at the code to make sure you understand the basic structure.

(a) As a first step, we will consider the Euclidean metric to determine the closest vertex to a new sample. One thing that you need to be careful about is that the θ variable (the first coordinate of our state) wraps around. In particular, θ will lie between $-\frac{\pi}{2}$ and $\frac{3\pi}{2}$. Thus, the angles $-\frac{\pi}{2}$ and $\frac{3\pi}{2}$ are in fact the same. Given this representation, what is the Euclidean distance between the states $[-1; 4]$ and $[4; -3]$?

Answer: 7.1166

Explanation

The difference between the angle coordinates (taking wrapping into account) is: $\mathbf{mod}(4 - (-1) + \pi, 2\pi) - \pi \approx -1.2832$. The difference in the angular velocity coordinate is 7. Hence, the Euclidean distance is approximately 7.1166.

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

RRT with dynamic constraints (Part B)

0.0/15.0 points (ungraded)

(b) Next, we will implement the RRT algorithm using the Euclidean metric as our distance function. In order to do this, you will need to fill in the parts of the code that say "FILL ME IN" by implementing the following:

- Implement the function **closest_vert = closestVertexEuclidean(rrt_verts,xy)** that takes in a $2 \times K$ vector consisting of the current vertices of the RRT, along with a state xy and returns the vertex in the tree that is closest to xy.

- Implement the function **new_vert = extendEuclidean(closest_vert,xy)** that will perform the extend operation in the following manner. Discretize the range of control inputs $[-5, 5]$ (around 20 discrete samples should be sufficient) and choose the input u_0 that will result in the system moving the most in the direction of the sample point xy when started from the state closest_vert (there are many ways to make this choice and we leave this to you to explore). Simulate the system (using ode45 for example) for a small time interval (no more than 0.1 seconds) using this constant control input to obtain new_vert. **Be sure to correctly wrap the angle coordinate. Also make sure to implement the input saturations in your simulation.**

Copy the $K \times 2$ array "rrt_verts_grade" you obtained below. The first row should be the start state and the last row should be (close to) the goal state.

```
1 rrt_verts_grade = [];
2
```

Unanswered

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for finding closest vertex
function closest_vert = closestVertexEuclidean(rrt_verts,xy)

% Compute distances (make sure to wrap the angle coordinate)
th_dists = diff(unwrap([repmat(xy(1),1,size(rrt_verts,2));rrt_verts(1,:)]));
thdot_dists = rrt_verts(2,:) - xy(2);

% Compute distances in a vectorized way
dists = sqrt(th_dists.^2 + thdot_dists.^2);

% Find minimizer
[~,minind] = min(dists);

% Closest vertex
closest_vert = rrt_verts(:,minind);



end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for extending tree
function xnew = extendEuclidean(closest_vert,xy)

% Find u0 that will move system towards xy
u0s = linspace(-5,5,20);

th_diff = diff(unwrap([closest_vert(1);xy(1)]));
cosangle = -Inf;
for k = 1:length(u0s)
    xdot_u0 = [closest_vert(2); (u0s(k) -9.81*1*sin(closest_vert(1)) - 0.1*closest_vert(2))];
    cosangle_u0 = dot([th_diff;xy(2)-closest_vert(2)],xdot_u0/norm(xdot_u0));
    if cosangle_u0 > cosangle
        cosangle = cosangle_u0;
        u0 = u0s(k);
    end
end

% Simulate policy
[~,Y] = ode45(@(t,x)pendulumDynamics(t,x,[0 0],xy,u0),[0 0.1],closest_vert);

% Extended state
xnew = Y(end,:)';

% Wrap state
```

```
xnew(1) = mod(xnew(1)+pi/2,2*pi) - pi/2;

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pendulum dynamics
function xdot = pendulumDynamics(t,x,K,x0,u0)

xdot = zeros(2,1);
xdot(1) = x(2);

u = -K*(x-x0)+u0;

if u > 5
    u = 5;
end
if u < -5
    u = -5;
end

g = 9.81;
b = 0.1;
xdot(2) = (u -g*1*sin(x(1)) - b*x(2));

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rrt_dynamic.m code

%% Code setup (Do not modify, but please read) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Bounds on world
world_bounds_th = [-pi/2,(3/2)*pi];
world_bounds_thdot = [-10,10];

% Start and goal positions
figure(1);
xy_start = [0;0]; plot(xy_start(1),xy_start(2),'bo','MarkerFaceColor','b','MarkerSize',10);
xy_goal = [pi;0]; plot(xy_goal(1),xy_goal(2),'go','MarkerFaceColor','g','MarkerSize',10); drawnc

% Initialize RRT. The RRT will be represented as a 2 x N list of points. So
% each column represents a vertex of the tree.
rrt_verts = zeros(2,1000);
rrt_verts(:,1) = xy_start;
N = 1;
nearGoal = false; % This will be set to true if goal has been reached
```

```
minDistGoal = 0.25; % This is the convergence criterion. We will declare
                    % success when the tree reaches within 0.25 in distance
                    % from the goal. DO NOT MODIFY.

% Keep track of where a node came from in order to reconstruct path
came_from = 1;

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Choose one of these methods
method = 'euclidean'; % Euclidean distance metric (part b of problem)
% method = 'lqr'; % LQR distance metric (part d of problem)
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


figure(1); hold on;
axis([world_bounds_th, world_bounds_thdot]);
 hxy = plot(0,0,'ro');

 iter = 1;
% RRT algorithm
while ~nearGoal
    iter = iter + 1;

    % Sample point
    rnd = rand(1);
    % With probability 0.05, sample the goal. This promotes movement to the
    % goal.
    if rnd < 0.05
        xy = xy_goal;
    else
        % Sample from space with probability 0.95
        xs = (world_bounds_th(2) - world_bounds_th(1))*rand(1) + world_bounds_th(1);
        ys = (world_bounds_thdot(2) - world_bounds_thdot(1))*rand(1) + world_bounds_thdot(1);
        xy = [xs;ys];
    end

    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if strcmp(method, 'euclidean')
        closest_vert = closestVertexEuclidean(rrt_verts(:,1:N),xy); % Write this function
    elseif strcmp(method, 'lqr')
        [closest_vert,K,closestInd] = closestVertexLQR(rrt_verts(:,1:N),xy); % Write this functi
        % To know where we must go, we must remember where we came from.
        came_from(iter) = closestInd;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if strcmp(method, 'euclidean')
        new_vert = extendEuclidean(closest_vert,xy); % Write this function
    else
        new_vert = extendLQR(closest_vert,xy,K); % Write this function
    end

    delete(hxy);
    figure(1);
    hxy = plot(xy(1),xy(2),'r.');axis([world_bounds_th, world_bounds_thdot]);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Plot extension (Comment the next few lines out if you want your code to
    % run a bit quicker. The plotting is useful for debugging though.)
    figure(1)
    hold on
    plot(new_vert(1),new_vert(2),'bo','MarkerFaceColor','b','MarkerSize',5);
    % Plot line (but only if we are not wrapping to the other side of the
    % plot)
    if abs(closest_vert(1) - new_vert(1)) < 0.75*(2*pi)
        line([closest_vert(1),new_vert(1)],[closest_vert(2),new_vert(2)]);
    end
    axis([world_bounds_th, world_bounds_thdot]);


    %% DO NOT MODIFY THIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If it is collision free, add it to tree
    N = N+1;
    if N > size(rrt_verts,2)
        rrt_verts = [rrt_verts zeros(size(rrt_verts))];
    end
    rrt_verts(:,N) = new_vert;

    % Check if we have reached goal
    if norm(xy_goal-new_vert) < minDistGoal
        break;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


end

% Plot vertices in RRT
hold on;
```

```
plot(rrt_verts(1,:),rrt_verts(2,:),'bo','MarkerFaceColor','b','MarkerSize',5);

%% Submit rrt_verts_grade for grading %%%%
format long
rrt_verts_grade = rrt_verts(:,1:N)'
clipboard('copy',rrt_verts_grade);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

RRT with dynamic constraints (Part C)

0.0/10.0 points (ungraded)

(c) If you run your code from part (b) a few times, you should see that although the RRT usually finds a feasible path to the goal, it can be frustratingly slow. This is in large part due to the fact that the Euclidean distance is not a very good metric for our problem. We will now consider an alternative based on LQR that typically works much better.

Given a nominal state $\boldsymbol{x_0}$ and control input $\boldsymbol{u_0}$, we can linearize the dynamics $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ about this point:

$$\dot{\boldsymbol{x}} \approx \boldsymbol{f}(\boldsymbol{x_0}, \boldsymbol{u_0}) + \frac{\partial \boldsymbol{f}(\boldsymbol{x_0}, \boldsymbol{u_0})}{\partial \boldsymbol{x}}(\boldsymbol{x} - \boldsymbol{x_0}) + \frac{\partial \boldsymbol{f}(\boldsymbol{x_0}, \boldsymbol{u_0})}{\partial \boldsymbol{u}}(\boldsymbol{u} - \boldsymbol{u_0}).$$

We can then define a change of variables $\bar{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x_0}$ and $\bar{\boldsymbol{u}} = \boldsymbol{u} - \boldsymbol{u_0}$. Then we have:

$$\dot{\boldsymbol{x}} \approx \boldsymbol{f}(\boldsymbol{x_0}, \boldsymbol{u_0}) + \boldsymbol{A}(\boldsymbol{x_0}, \boldsymbol{u_0})\,\bar{\boldsymbol{x}} + \boldsymbol{B}(\boldsymbol{x_0}, \boldsymbol{u_0})\,\bar{\boldsymbol{u}},$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are the partial derivatives of $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ with respect to $\boldsymbol{x}$ and $\boldsymbol{u}$ respectively.

Given a quadratic cost function defined by state and action cost matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, we can solve a LQR problem ([K,S] = lqr(A,B,Q,R) in Matlab). Then the locally optimal policy is given by $\boldsymbol{u}^{\star}(\bar{\boldsymbol{x}}) = -\boldsymbol{K}\bar{\boldsymbol{x}} + \boldsymbol{u_0}$. The metric we will use to determine how close a point $\boldsymbol{x}$ is to $\boldsymbol{x_0}$ is then given by $(\boldsymbol{x} - \boldsymbol{x_0})\,\boldsymbol{S}\,(\boldsymbol{x} - \boldsymbol{x_0})$.

For the pendulum, given $x_0 = [\pi - 0.1; 2.0]$ and $u_0 = 0$, what is the "distance" (evaluated by the metric just described) between x_0 and $x = [\pi + 0.2; -5.0]$? (Use $Q = I_{2\times 2}$ (the identity matrix) and $R = 0.1$).

Answer: 26.4296

Explanation
The difference between the angle coordinates (taking wrapping into account) is 0.3. The difference in the angular velocity coordinate is -7. The matrices of partial derivatives are given by $A = [0, 1; -9.81\cos(\pi - 0.1), -0.1]$ and $B = [0; 1]$. From lqr(A,B,eye(2),0.1), we get $S \approx [7.3567, 2.0021; 2.0021, 0.6975]$. Thus, the distance is $[0.3, -7] * S * [0.3; -7] \approx 26.4296.$

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

RRT with dynamic constraints (Part D)

0.0/20.0 points (ungraded)
(d) Next, we will use the LQR metric from (c) in our RRT algorithm. In order to do this, you will need to implement the following:

- Implement the function **[closest_vert,K] = closestVertexLQR(rrt_verts,xy)** that takes in a $2 \times M$ vector consisting of the current vertices of the RRT, along with a state xy and returns the vertex in the tree that is closest to xy (as measured by the LQR metric), along with the gain matrix of the LQR policy. **Make sure to handle the wrapping of the angle variable correctly.**

- Implement the function **new_vert = extendLQR(closest_vert,xy,K)** that will perform the extend operation by applying the LQR policy starting from the state closest_vert for a short time interval (no more than 0.1 seconds). Here, you should use $x_0 =$ xy and $u_0 = 0$ to find the LQR policy. You can use ode45 again for the integration. Again, make sure to correctly wrap the angle coordinate. **Also make sure to implement the input saturations in your simulation.**

- Modify your code to output the path from start to goal found by the RRT. This should be a $L \times 2$ array. Given that this array is called "xpath", you can use the code provided here to visualize the path using the PendulumVisualizer in the examples/Pendulum folder in drake (you'll need to be in this folder to run this). Once you are satisfied that the path has been

correctly obtained, type in your xpath below. **Make sure that it is a $L \times 2$ array. The first row should be the start state and the last row should be (close to) the goal state. Do not type in rrt_verts_grade.**

```
xpath = [];

```

Unanswered

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for finding closest vertex using LQR "metric"
function [closest_vert,K,minind] = closestVertexLQR(rrt_verts,xy)

% Linearize p
A = [0 1; -9.81*cos(xy(1)), -0.1];
B = [0; 1];

% LQR
Q = diag([10 1]);
R = 1;

[K,S] = lqr(full(A),full(B),Q,R);

% Compute distances (make sure to wrap the angle coordinate)
th_dists = diff(unwrap([repmat(xy(1),1,size(rrt_verts,2)); rrt_verts(1,:)]));
% th_dists = rrt_verts(1,:) - repmat(xy(1),1,size(rrt_verts,2));
thdot_dists = rrt_verts(2,:) - xy(2);

% Compute distances in a vectorized way (this is not the most efficient
% vectorization, but is good enough for our purposes).
dists = diag([th_dists;thdot_dists]'*S*[th_dists;thdot_dists]);

% Find minimizer
[~,minind] = min(dists);

% Closest vertex
closest_vert = rrt_verts(:,minind);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for performing extension
function xnew = extendLQR(closest_vert,xy,K)

% Simulate LQR policy
[~,Y] = ode45(@(t,x)pendulumDynamics(t,x,K,xy,0),[0 0.1],closest_vert);

% Extended state
xnew = Y(end,:)';

% Wrap state
xnew(1) = mod(xnew(1)+pi/2,2*pi) - pi/2;

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pendulum dynamics
function xdot = pendulumDynamics(t,x,K,x0,u0)

xdot = zeros(2,1);
xdot(1) = x(2);

u = -K*(x-x0)+u0;

if u > 5
    u = 5;
end
if u < -5
    u = -5;
end

g = 9.81;
b = 0.1;
xdot(2) = (u -g*1*sin(x(1)) - b*x(2));

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rrt_dynamic.m with complete code

%% Code setup (Do not modify, but please read) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Bounds on world
world_bounds_th = [-pi/2,(3/2)*pi];
world_bounds_thdot = [-10,10];

% Start and goal positions
figure(1);
xy_start = [0;0]; plot(xy_start(1),xy_start(2),'bo','MarkerFaceColor','b','MarkerSize',10);
xy_goal = [pi;0]; plot(xy_goal(1),xy_goal(2),'go','MarkerFaceColor','g','MarkerSize',10); drawnc

% Initialize RRT. The RRT will be represented as a 2 x N list of points. So
% each column represents a vertex of the tree.
rrt_verts = zeros(2,1000);
rrt_verts(:,1) = xy_start;
N = 1;
nearGoal = false; % This will be set to true if goal has been reached
minDistGoal = 0.25; % This is the convergence criterion. We will declare
                    % success when the tree reaches within 0.25 in distance
                    % from the goal. DO NOT MODIFY.
```

```
% Keep track of where a node came from in order to reconstruct path
came_from = 1;

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Choose one of these methods
% method = 'euclidean'; % Euclidean distance metric (part b of problem)
method = 'lqr'; % LQR distance metric (part d of problem)
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


figure(1); hold on;
axis([world_bounds_th, world_bounds_thdot]);
 hxy = plot(0,0,'ro');

 iter = 1;
% RRT algorithm
while ~nearGoal
    iter = iter + 1;

    % Sample point
    rnd = rand(1);
    % With probability 0.05, sample the goal. This promotes movement to the
    % goal.
    if rnd < 0.05
        xy = xy_goal;
    else
        % Sample from space with probability 0.95
        xs = (world_bounds_th(2) - world_bounds_th(1))*rand(1) + world_bounds_th(1);
        ys = (world_bounds_thdot(2) - world_bounds_thdot(1))*rand(1) + world_bounds_thdot(1);
        xy = [xs;ys];
    end

    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if strcmp(method, 'euclidean')
        closest_vert = closestVertexEuclidean(rrt_verts(:,1:N),xy); % Write this function
    elseif strcmp(method, 'lqr')
        [closest_vert,K,closestInd] = closestVertexLQR(rrt_verts(:,1:N),xy); % Write this functi
        % To know where we must go, we must remember where we came from.
        came_from(iter) = closestInd;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if strcmp(method, 'euclidean')
        new_vert = extendEuclidean(closest_vert,xy); % Write this function
    else
```

```
        new_vert = extendLQR(closest_vert,xy,K); % Write this function
    end

    delete(hxy);
    figure(1);
    hxy = plot(xy(1),xy(2),'r.');axis([world_bounds_th, world_bounds_thdot]);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Plot extension (Comment the next few lines out if you want your code to
    % run a bit quicker. The plotting is useful for debugging though.)
    figure(1)
    hold on
    plot(new_vert(1),new_vert(2),'bo','MarkerFaceColor','b','MarkerSize',5);
    % Plot line (but only if we are not wrapping to the other side of the
    % plot)
    if abs(closest_vert(1) - new_vert(1)) < 0.75*(2*pi)
        line([closest_vert(1),new_vert(1)],[closest_vert(2),new_vert(2)]);
    end
    axis([world_bounds_th, world_bounds_thdot]);


    %% DO NOT MODIFY THIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If it is collision free, add it to tree
    N = N+1;
    if N > size(rrt_verts,2)
        rrt_verts = [rrt_verts zeros(size(rrt_verts))];
    end
    rrt_verts(:,N) = new_vert;

    % Check if we have reached goal
    if norm(xy_goal-new_vert) < minDistGoal
        break;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


end

% Plot vertices in RRT
hold on;
plot(rrt_verts(1,:),rrt_verts(2,:),'bo','MarkerFaceColor','b','MarkerSize',5);

%% Submit rrt_verts_grade for grading %%%%
format long
```

```
rrt_verts_grade = rrt_verts(:,1:N)'
clipboard('copy',rrt_verts_grade);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Reconstruct path
reachedStart = false;
indnext = length(rrt_verts_grade);
path_inds = indnext;
while ~reachedStart
    indnext = came_from(indnext);
    path_inds = [path_inds,indnext];
    if indnext == 1
        reachedStart = true;
    end
end

xpath = rrt_verts(:,fliplr(path_inds));
xpath = xpath';

%% Visualize path (if we're in the drake/examples/Pendulum folder)
if exist('PendulumVisualizer', 'file')
  v = PendulumVisualizer();
  xp1 = unwrap(xpath(:,1)+2*pi);
  xtraj = PPTrajectory(spline(0:0.1:0.1*(length(xpath)-1),[xp1,xpath(:,2)]'));
  xtraj = xtraj.setOutputFrame(v.getInputFrame);
  v.playback(xtraj);
end
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Model Predictive Control

Model Predictive Control, Part 1

0.0/10.0 points (graded)

Given a discrete linear system, corresponding to a discretization of a second order system,

$$x[k+1] = \begin{bmatrix} 1 & 0 & .1 & 0 \\ 0 & 1 & 0 & .1 \\ 2 & -2 & 1.2 & 0 \\ 0 & 1.5 & -1 & .7 \end{bmatrix} x[k] + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & -2 \\ 0 & 1 \end{bmatrix} u[k]$$

Write a quadratic program (QP) to do model predictive control. From an initial state $x[0] = x_0$, find a sequence of inputs that minimizes the cost function: *[Math Processing Error]*

$$\sum_{k=1}^{N} ||u[k-1]||^2 + .01||x[k]||^2$$

such that the state is driven to the origin, $x[N] = 0^{4\times 1}$. Please use the QP format given--note that your solution will be graded on correctness of the control sequence. Ensure that the matrix "u" in your solution is $2 \times N$, where $u[k]$=u(:,k+1). You may implement this as a shooting or transcription type approach.

To avoid confusion, write the QP in the same format as MATLAB's quadprog:

$$\begin{aligned} \min .5 z^T H z + f^T z \\ \text{s.t.} Az \leq b \\ Bz = c \end{aligned}$$

```
% PROBLEM SETUP, DO NOT CHANGE
A_sys = [1 0 .1 0;0 1 0 .1; 2 -2 1.2 0; 0 1.5 -1 .7];
B_sys = [0 0;0 0;1 -2;0 1];
N = 10;
x0 = 10*randn(4,1);
```

```
6
7         % QP SETUP HERE
8         H =
9         f =
10        A =
11        b =
12        B =
13        c =
14
15        % SOLVE QP
```

Unanswered

```
    % Take the transcription based approach
            % order z = [u0;x1;u1;x2;...;uN-1;xN]
            num_vars = 6*N;
            H = zeros(num_vars);
            f = zeros(num_vars,1);
            A = zeros(0,num_vars);
            b = [];
            B = zeros(0,num_vars);
            c = [];

            % dynamic constraints
            for k=1:N,
              H((1:2)+6*(k-1),(1:2)+6*(k-1)) = eye(2);
              H((3:6)+6*(k-1),(3:6)+6*(k-1)) = .01*eye(4);
              B_dyn = zeros(4,num_vars);
              if k==1,
                    B_dyn(:,1:6) = [B_sys -eye(4)];
                    c = [c;-A_sys*x0];
              else
                start_ind = 6*(k-1) - 4;
                    B_dyn(:,start_ind + (1:10)) = [A_sys B_sys -eye(4)];
                    c = [c;zeros(4,1)];
          end
              B = [B;B_dyn];
            end

            % goal constraint
            B = [B;zeros(4,num_vars-4) eye(4)];
            c = [c;zeros(4,1)];

options = optimset('Display','Off');
            [z,fval,exitflag] = quadprog(H,f,A,b,B,c,[],[],[],options);

            % extract solution
            z_reshape = reshape(z,6,N);
            x_sol = z_reshape(3:end,:);
            u = z_reshape(1:2,:);
```

Run Code

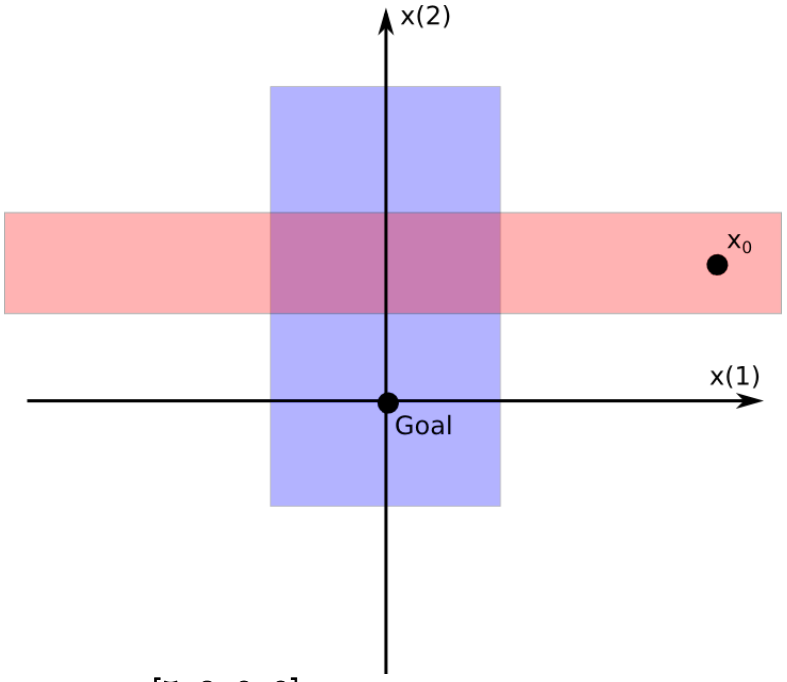
Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Model Predictive Control, Part 2

0.0/10.0 points (graded)

Let's make the previous problem a a little more interesting. Suppose now that we have to stay inside a pair of intersecting corridors. In particular, for all $x\,[k]$, we must have $x\,(2) \in [2,4]$ **OR** $x\,(1) \in [-2,2]$, referring to the second and first element of the vector x respectively. Note, we are only enforcing this constraint at the discrete points, not between them!



For $x_0 = [5;3;0;0]$, modify the previous program so that the first condition above holds for $k = 1,\ldots,m$ and the second condition holds for $k = m,\ldots,N$. Note the overlap on index m! Run your program for all possible values of m, and select the optimal u, as in Part 1, and the optimal point m^*.

Note that the program may not always be feasible! Be sure to check the solutions for feasibility.

```
% PROBLEM SETUP, DO NOT CHANGE
A_sys = [1 0 .1 0;0 1 0 .1; 2 -2 1.2 0; 0 1.5 -1 .7];
    B_sys = [0 0;0 0;1 -2;0 1];
    N = 10;
    x0 = [5;3;0;0];

m_star =
u =
```

```
9
```

Unanswered

```
  % setup from part 1
  num_vars = 6*N;
  H = zeros(num_vars);
  f = zeros(num_vars,1);
  A = zeros(0,num_vars);
  b = [];
  B = zeros(0,num_vars);
  c = [];

  % dynamic constraints
  for k=1:N,
    H((1:2)+6*(k-1),(1:2)+6*(k-1)) = eye(2);
    H((3:6)+6*(k-1),(3:6)+6*(k-1)) = .01*eye(4);
    B_dyn = zeros(4,num_vars);
    if k==1,
          B_dyn(:,1:6) = [B_sys -eye(4)];
          c = [c;-A_sys*x0];
    else
      start_ind = 6*(k-1) - 4;
          B_dyn(:,start_ind + (1:10)) = [A_sys B_sys -eye(4)];
          c = [c;zeros(4,1)];
end
    B = [B;B_dyn];
  end

  % goal constraint
  B = [B;zeros(4,num_vars-4) eye(4)];
  c = [c;zeros(4,1)];

  opt_cost = inf;
  opt_m = [];
  opt_u = [];
  for m=1:N,
          A = zeros(0,num_vars);
          b = [];
          for k=1:m,
                  A_row = zeros(1,num_vars);
                  A_row(6*(k-1)+4) = 1;
                  A = [A;A_row;-A_row];
                  b = [b;4;-2];
          end
          for k=m:N,
                  A_row = zeros(1,num_vars);
                  A_row(6*(k-1)+3) = 1;
                  A = [A;A_row;-A_row];
                  b = [b;2;2];
          end
```

```
    options = optimset('Display','Off');
                    [z,fval,exitflag] = quadprog(H,f,A,b,B,c,[],[],[],options);

                    if exitflag == 1,
                            if lt(fval,opt_cost)
                                    opt_cost = fval;
                                    opt_m = m;
                                    % extract solution
                                    z_reshape = reshape(z,6,N);
                                    x_sol = z_reshape(3:end,:);
                            u_sol = z_reshape(1:2,:);
                            end
                    end
            end

u = u_sol;
m_star = opt_m;
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Model Predictive Control, Part 3

0.0/7.0 points (graded)

For the QP from **Part 1**, which of the following statements are true?

- The program is guaranteed to return a control sequence $u[k]$. ✔
- The program has a unique optimal control sequence $u[k]$. ✔
- The first control element, $u[0]$ is *continuous* with respect to the initial state $x[0]$. ✔

Submit You have used 0 of 1 attempt

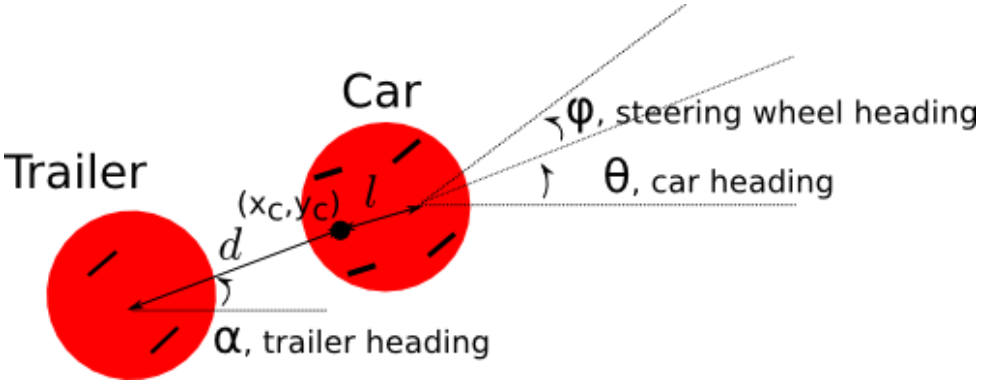
Answers are displayed within the problem

Differential Flatness

Differential Flatness, Part 1

0.0/10.0 points (graded)

Car with trailer



Suppose we have a car with a trailer attached. For simplicity, model both car and trailer as circular objects. Letting the state be the position of the car, and absolute angles of both car and trailer (as shown). Using a simple vehicle model, where the control inputs are the speed, u_1, and the angular rate of the front wheels, u_2, the dynamics are:

$$\dot{x} = \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} \cos(\theta)\, u_1 \\ \sin(\theta)\, u_1 \\ u_2 \\ l^{-1} \tan(\phi)\, u_1 \\ d^{-1} \sin(\theta - \alpha)\, u_1 \end{bmatrix}$$

It turns out that this system is differentially flat, even with any number of trailers strung out behind the car! Take the flat output y to be the position of the trailer:

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_c - d\cos(\alpha) \\ y_c - d\sin(\alpha) \end{bmatrix}$$

If y is truly a flat output, which of the following statements is true?

- (A) We can write $x = f(y)$.
- (B) We can write $u = g(y)$.
- (C) We can write $x = f(y, \dot{y}, \ddot{y}, \dots)$.
- (D) We can write $u = g(y, \dot{y}, \ddot{y}, \dots)$.
- (E) A and B.
- (F) C and D. ✔

Find $\tan(\alpha)$ in terms of $y_1, y_2, \dot{y}_1, \dot{y}_2$. For your answer, write $v_i = \dot{y}_i$.

Answer: v_2/v_1

Ignoring singularities, we now have a function for α. Verify for yourself that, using α, you can easily find functions x_c and y_c, which can be differentiated and then used to find an expression for θ and so on.

Submit You have used 0 of 1 attempt

Answers are displayed within the problem

Differential Flatness, Part 2

0.0/20.0 points (graded)

QP Control of the Car

Suppose we wish to pose a planning problem for the car with a trailer. For a fixed duration T, we would like to get from an initial state x_0 and bring the trailer to a desired location y_g.

Suppose also that we choose to parameterize the path in the output space as:

$$y_1 = \sum_{i=0}^{N} a_i t^i$$

$$y_2 = \sum_{i=0}^{N} b_i t^i$$

For some sufficiently large N. For decision variables $a_0, \ldots, a_N, b_0, \ldots, b_N$, and $N = 4$, write the minimum jerk QP:

[Math Processing Error]

$$\begin{aligned} \min & \int_0^T ||y^{(3)}||^2 dt \\ \text{s.t.} & y(0) = y_0 \\ & \dot{y}(0) = \dot{y}_0 \\ & \ddot{y}(0) = \ddot{y}_0 \\ & y(T) = y_g \end{aligned}$$

Note that, for now, we've written the initial state constraint on the output. To avoid confusion, write the QP in the same format as MATLAB's quadprog:

$$\begin{aligned} \min & .5 z^T H z + f^T z \\ \text{s.t.} & Az \leq b \\ & Bz = c \end{aligned}$$

```
% PROBLEM SETUP, DO NOT CHANGE
N = 4;
T = 5;
y_0 = randn(2,1);
yd_0 = randn(2,1);
ydd_0 = randn(2,1);
y_g = 2 + randn(2,1);

% QP SETUP HERE
H =
f =
A =
b =
B =
c =

```

Unanswered

```
% first, solve for the cost
% substituting the polynomials into the term above
% we get the integral from 0 to T of:
% 36(a_3^2 + 16*a_4^2*t^2 + 8*a_3*a_4*t)
% with a similar term for b
% Then, the cost is: 36(T*a_3^2 + 16/3*T^3*a_4^2 + 4*T^2*a_3*a_4)
% we can ignore constant factors (the 36), and then:
H = zeros(10);
H(4,4) = T;
H(5,5) = 16/3*T^3;
H(4,5) = 2*T^2;
H(5,4) = 2*T^2;
H(9,9) = T;
H(10,10) = 16/3*T^3;
H(9,10) = 2*T^2;
H(10,9) = 2*T^2;

f = zeros(10,1);

A = zeros(0,10);
b = [];

% initial state
B = [diag([1;1;2]) zeros(3,7); zeros(3,5) diag([1;1;2]) zeros(3,2)];
c = [y_0(1);yd_0(1);ydd_0(1);y_0(2);yd_0(2);ydd_0(2)];

% goal
B = [B;[1 T T^2 T^3 T^4 zeros(1,5);zeros(1,5) 1 T T^2 T^3 T^4]];
c = [c;y_g];
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

Differential Flatness, Part 3

0.0/18.0 points (graded)

QP Possibilities

This last question will explore what we can and cannot easily do with QP optimization. All of these questions consider adding constraints *only* to the problem above, and the answers may not be obvious at first! You are encouraged to carefully think these questions through before answering.

In writing down additional constraints, it's possible that the program becomes overconstrained, and infeasible. For now, ignore this question and focus on whether or not we can pose the problem.

In the previous formulation, we constrainted $\boldsymbol{y}\left(0\right), \dot{\boldsymbol{y}}\left(0\right), \ddot{\boldsymbol{y}}\left(0\right)$ instead of $\boldsymbol{x}\left(0\right)$. Did we need to do that? For a fixed $\boldsymbol{x_0}$, could we have written the constraint $\boldsymbol{x}\left(0\right) = \boldsymbol{x_0}$ as a *linear function* of the decision variables?

- Yes, there is such a linear constraint ✔
- No, that constraint is not linear.

What about constraining the final state, $\boldsymbol{x}\left(T\right)$, assuming such a state is not inconsistent with the goal state $\boldsymbol{y}\left(T\right)$?

- Yes, there is such a linear constraint ✔
- No, that constraint is not linear

Corridor Constraints

For the last four questions, we wish to stay within a physical corridor (e.g. an indoor hallway or a lane on a highway), and suppose the corridor is a convex polytope in the physical plane.

Suppose we had the task of ensuring that the trailer stayed within the corridor for all $\boldsymbol{t} \in \left[\boldsymbol{0}, \boldsymbol{T}\right]$. Could we write that constraint as a *linear function* of the decision variables?

- Yes, there is such a linear constraint
- No, that constraint is not linear ✔

For some fixed integer M, suppose we had the task of ensuring that the trailer stayed within the corridor for all $t \in \{0, \frac{T}{M}, \frac{2T}{M}, \ldots T\}$. Could we write that constraint as a *linear function* of the decision variables?

- Yes, there is such a linear constraint ✔
- No, that constraint is not linear

Suppose we had the task of ensuring that the trailer *and car* stayed within the corridor for all $t \in [0, T]$. Could we write that constraint as a *linear function* of the decision variables?

- Yes, there is such a linear constraint
- No, that constraint is not linear ✔

For some fixed integer M, suppose we had the task of ensuring that the trailer *and car* stayed within the corridor for all $t \in \{0, \frac{T}{M}, \frac{2T}{M}, \ldots T\}$. Could we write that constraint as a *linear function* of the decision variables?

- Yes, there is such a linear constraint
- No, that constraint is not linear ✔

Submit You have used 0 of 1 attempt

Answers are displayed within the problem