Spring 2022

Stanford
AA 203: Optimal and Learning-based Control
Homework #1, due April 18 by 11:59 pm

Pei-Chen Wu pcwu1023

Collaborator: Kevin Lee and Albert Chan

Problem 1: Backstepping

Consider the strict-feedback system

$$\begin{aligned}\dot{x} &= f(x) + B(x)z \\ \dot{z} &= u\end{aligned},$$

with $x \in \mathbb{R}^n$ and $z, u \in \mathbb{R}^m$, where $f : \mathbb{R}^n \to \mathbb{R}^n$ and $B : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are known smooth functions, and $f(0) = 0$.

Suppose the subsystem $\dot{x} = f(x) + B(x)z$ can be stabilized by a smooth feedback law $z = \phi_0(x)$ with $\phi_0(0) = 0$, i.e., the closed-loop system $\dot{x} = f(x) + B(x)\phi_0(x)$ is *globally asymptotically stable* with respect to the origin $x = 0$. Moreover, suppose we know a smooth, positive-definite, radially unbounded Lyapunov function $V_0 : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ and positive definite function $\rho : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ satisfying

$$\dot{V}_0(x) = \nabla V_0(x)^\mathsf{T}(f(x) + B(x)\phi_0(x)) \leq -\rho(x),$$

for all $x \in \mathbb{R}^n$.

We now consider the entire (x, z)-system, which we can only control through $u \in \mathbb{R}^m$. We want to use our knowledge of a stabilizing controller for the inner x-dynamics and the strict-feedback form of the (x, z)-dynamics to "back out" a stabilizing controller for the entire system.

Use the Lyapunov candidate function

$$V_1(x, z) = V_0(x) + \frac{1}{2}\|z - \phi_0(x)\|_2^2$$

to find a stabilizing controller $u = \phi_1(x, z)$ for some function $\phi_1 : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$ that ensures $(x, z) \to (0, 0)$. Notice that V_1 comprises the "inner" Lyapunov function V_0 and a penalty term for the difference between z and the value of the "inner" stabilizing control. Explicitly derive the function ϕ_1 and rigorously describe why it stabilizes the (x, z)-system using Lyapunov theory (i.e., prove $V_1(x, z)$ is positive-definite and radially unbounded, and $\dot{V}(x, z)$ is negative-definite along trajectories of the (x, z)-subsystem in closed-loop with $u = \phi_1(x, z)$).

ANSWER:

- Prove $V_1(x, z)$ is positive-definite and radially unbounded.

$$V_1(x, z) = V_0(x) + \frac{1}{2}\|z - \phi_0(x)\|_2^2$$

From problem statement, we can know that $V_0 : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is positive-definite, radially unbounded Lyapunov function and the second term $\frac{1}{2}\|z - \phi_0(x)\|_2^2$ is guaranteed to be positive.

Therefore, $V_1(x,z)$ is promised to be positive-definite and radially unbounded.

- Find a stabilizing controller $u = \phi_1(x,z)$ which can make $\dot{V}(x,z)$ negative-definite along trajectories of the (x,z)-subsystem in closed-loop.

$$\dot{V}_1(x,z) = \nabla_x V_1(x)^{\mathsf{T}} \dot{x} + \nabla_z V_1(x)^{\mathsf{T}} \dot{z}$$

$$\dot{V}_1(x,z) = \nabla_x V_1(x)^{\mathsf{T}} \dot{x} + \nabla_z V_1(x)^{\mathsf{T}} u$$

$$\nabla_x V_1 = \nabla_x V_0 - \nabla_x \phi_0 (z - \phi_0)$$

$$\nabla_z V_1 = (z - \phi_0)$$

$$\dot{V}_1 = \nabla_x V_1(x)^{\mathsf{T}} (f + Bz) + \nabla_z V_1(x)^{\mathsf{T}} u$$

$$\dot{V}_1 = \nabla_x V_1(x)^{\mathsf{T}} f + \nabla_x V_1(x)^{\mathsf{T}} Bz + \nabla_z V_1(x)^{\mathsf{T}} u$$

$$\dot{V}_1 = \nabla_x V_0{}^{\mathsf{T}} f - \nabla_x \phi_0 (z-\phi_0)^{\mathsf{T}} f + \nabla_x V_0{}^{\mathsf{T}} Bz - \nabla_x \phi_0 (z-\phi_0)^{\mathsf{T}} Bz + (z-\phi_0)^{\mathsf{T}} u + \nabla_x V_0{}^{\mathsf{T}} B\phi_0 - \nabla_x V_0{}^{\mathsf{T}} B\phi_0$$

$$\dot{V}_1 = \nabla_x V_0{}^{\mathsf{T}} (f + B\phi_0) - (z - \phi_0)^{\mathsf{T}} \Big(\nabla_x \phi_0 (f + Bz) - \nabla_x V_0^{\mathsf{T}} B - u \Big)$$

Since $\dot{V}_0 = \nabla_x V_0{}^{\mathsf{T}} (f + B\phi_0) \le -\rho(x)$.

$$\dot{V}_1 \le -\rho - (z - \phi_0)^{\mathsf{T}} \Big(\nabla_x \phi_0 (f + Bz) - \nabla_x V_0^{\mathsf{T}} B - u \Big)$$

$\nabla_x \phi_0 (f + Bz) = \nabla_x \phi_0 \dot{x} = \dot{\phi}_0$

$$\dot{V}_1 \le -\rho - (z - \phi_0)^{\mathsf{T}} \Big(\dot{\phi}_0 - \nabla_x V_0^{\mathsf{T}} B - u \Big)$$

We can choose $u = -(z - \phi_0) + \dot{\phi}_0 - \nabla_x V_0^{\mathsf{T}} B$, therefore, we can have a nice second term $\|z - \phi_0(x)\|_2^2$, which will make $\dot{V}_1$ to be negative-definite along trajectories of the (x,z)-subsystem in closed-loop with $u = \phi_1(x,y)$.

Problem 2: Model reference adaptive control

Consider the continuous-time system

$$\dot{y}(t) + \alpha y(t) = \beta u(t).$$

We want to control this system, but we do not know the true plant parameters $\alpha, \beta \in \mathbb{R}$. In this problem, we will use *direct Model-Reference Adaptive Control (MRAC)* to match the behaviour of the true plant with that of the reference model

$$\dot{y}_m(t) + \alpha_m y_m(t) = \beta_m r(t)$$

where $\alpha_m, \beta_m \in \mathbb{R}$ are *known* constant parameters, and $r(t)$ is a chosen *bounded* reference signal.

(a) Consider the control law

$$u(t) = k_r(t)r(t) + k_y(t)y(t)$$

where $k_r(t)$ and $k_y(t)$ are time-varying feedback gains. Write out the differential equation for the resulting closed-loop dynamics. Use this to verify that, if $y(0) = y_m(0)$ and we knew α and β, the constant control gains

$$k_r^* := \frac{\beta_m}{\beta}, \quad k_y^* := \frac{\alpha - \alpha_m}{\beta}$$

would make the true plant dynamics perfectly match the reference model.

ANSWER:

The differential equation for the resulting closed-loop dynamics:

$$\dot{y}(t) + (\alpha - \beta k_y(t))y(t) = \beta k_r(t)r(t)$$

$$\dot{y}(t) - \dot{y}_m(t) + (\alpha - \beta k_y(t))y(t) - \alpha_m y_m(t) = (\beta k_r(t) - \beta_m)r(t)$$

Plug in the control law $u(t) = k_r(t)r(t) + k_y(t)y(t)$ to the true plant dynamic equation and re-arrange it a little bit as the following.

$$\dot{y}(t) + (\alpha - \beta k_y(t))y(t) = \beta k_r(t)r(t)$$

Comparing with reference model,

$$\dot{y}_m(t) + \alpha_m y_m(t) = \beta_m r(t)$$

We can see that in order to match true plant dynamic with reference model, we can use $y(0) = y_m(0)$ and simply set $\alpha - \beta k_y(t) = \alpha_m$ and $\beta k_r(t) = \beta_m$. Re-arrange these two equations we can get $k_r^* := \frac{\beta_m}{\beta}$ and $k_y^* := \frac{\alpha - \alpha_m}{\beta}$.

(b) When we do not know α and β, we adaptively update our controller over time in response to measurements of $y(t)$. Specifically, we want an *adaptation law* for $k_r(t)$ and $k_y(t)$ to make $y(t)$ tend towards $y_m(t)$ asymptotically. For this, we define the tracking error $e(t) := y(t) - y_m(t)$ and the parameter errors

$$\delta_r(t) := k_r(t) - k_r^*, \quad \delta_y(t) := k_y(t) - k_y^*.$$

Determine a differential equation for e in terms of e, $\dot{e}$, y, r, δ_y, δ_r, and suitable constants.

ANSWER:

Re-arrange u to be $u(t) = \delta_r r(t) + \delta_y y(t) + k_r^* r(t) + k_y^* y(t)$. Plug in u to the true plant dynamic equation and minus the reference model dynamics, we will get the following equation.

$$\dot{y}(t) - \dot{y}_m(t) + \alpha y(t) - \alpha_m y_m(t) = \beta\delta_r r(t) + \beta\delta_y y(t) + \beta k_r^* r(t) + \beta k_y^* y(t) - \beta_m r(r)$$

Substitute $\dot{e}(t) := \dot{y}(t) - \dot{y_m}(t)$ and add $+\alpha_m y(t) - \alpha_m y(t)$ to the above equation.

$$\dot{e} + \alpha y(t) - \alpha_m y_m(t) + \alpha_m y(t) - \alpha_m y(t) = \beta\delta_r r(t) + \beta\delta_y y(t) + \beta k_r^* r(t) + \beta k_y^* y(t) - \beta_m r(r)$$

Since $\beta k_r^* r(t) = \beta_m r(r)$ and $(\alpha - \alpha_m)y(t) = \beta k_y^* y(t)$, we can cancel out these two terms and plug in $e(t) := y(t) - y_m(t)$. We can have the differential equation for e in terms of e, $\dot{e}$, y, r, δ_y, δ_r, and suitable constants.

$$\dot{e} + \alpha_m e(t) = \beta\delta_r r(t) + \beta\delta_y y(t)$$

We consider the adaptation law for k_r and k_y described by

$$\begin{aligned} \dot{k}_r(t) &= -\operatorname{sign}(\beta)\gamma e(t) r(t) \\ \dot{k}_y(t) &= -\operatorname{sign}(\beta)\gamma e(t) y(t) \end{aligned},$$

where $\gamma > 0$ is a chosen constant *adaptation gain*. Since we are adapting the gains k_r and k_y of our controller directly, rather than estimates of the system parameters α and β, this is a *direct* adaptation law. We must at least know the sign of β, which indicates in what direction the input $u(t)$ "pushes" the output $y(t)$. For example, when modeling a car, you could reasonably assume that an increased braking force slows down the car. To show that the tracking error and parameter errors are stabilized by our chosen control law and adaptation law, we use Lyapunov theory.

Theorem 1 (Lyapunov)**.** *Consider the continuous-time system $\dot{x} = f(x,t)$, where $x = 0$ is an equilibrium point, i.e., $f(0,t) \equiv 0$. Suppose there exists a continuously differentiable scalar function $V(x,t)$ such that V is positive-definite in x for each $t \geq 0$, and $\dot{V}$ is negative semi-definite in x for each $t \geq 0$. Then $x = 0$ is a stable point in the sense of Lyapunov, i.e., $\|x(t)\|_2$ remains bounded as long as $\|x(0)\|_2$ is bounded.*

(c) Consider the state $x := (e, \delta_r, \delta_y)$ and the Lyapunov function candidate

$$V(x) = \frac{1}{2}e^2 + \frac{|\beta|}{2\gamma}(\delta_r^2 + \delta_y^2).$$

Show that $\dot{V} = -\alpha_m e^2$. Based on Lyapunov theory, what can you say about $e(t)$, $\delta_r(t)$, and $\delta_y(t)$ for all $t \geq 0$ if $\alpha_m > 0$?

ANSWER:

Derivative of $V(x)$, we can get

$$\dot{V}(x) = e\dot{e} + \frac{|\beta|}{\gamma}(\dot{\delta}_r\delta_r + \dot{\delta}_y\delta_y).$$

Simply plug in $\dot{e} = -\alpha_m e(t) + \beta\delta_r r(t) + \beta\delta_y y(t)$ and the adaption law.

$$\dot{V}(x) = e\Big(- \alpha_m e(t) + \beta\delta_r r(t) + \beta\delta_y y(t)\Big) + \frac{|\beta|}{\gamma}\Big(- \operatorname{sign}(\beta)\gamma e(t)r(t)\delta_r - \operatorname{sign}(\beta)\gamma e(t)y(t)\delta_y\Big).$$

We can use $|\beta| \operatorname{sign}(\beta) = \beta$ and cancel out γ so we will have the following equation.

$$\dot{V}(x) = e\Big(- \alpha_m e(t) + \beta\delta_r r(t) + \beta\delta_y y(t)\Big) - \Big(\beta e(t)r(t)\delta_r + \beta e(t)y(t)\delta_y\Big).$$

We will have the resulting equation.

$$\dot{V} = -\alpha_m e^2(t)$$

Therefore, we can know that $V(x)$ is positive-definite from the problem definition, and we can know that $\dot{V}$ is negative semi-definite from the above equation. This means that the system is BIBO, we can be certain that $e(t)$, $\delta_r(t)$, and $\delta_y(t)$ are bounded for all $t \geq 0$ if $\alpha_m > 0$.

In general, adaptive controllers yield time-varying closed-loop dynamics, even for LTI systems. As a result, we require more mathematical machinery beyond basic Lyapunov theory to establish anything stronger than Lyapunov stability. To this end, we use Barbalat's Lemma.

Theorem 2 (Barbalat's Lemma)**.** *Suppose $g : \mathbb{R} \to \mathbb{R}$ is differentiable. If g has a finite limit as $t \to \infty$ and $\dot{g}$ is uniformly continuous, then* $\lim_{t\to\infty} \dot{g}(t) = 0$.

Boundedness of the derivative of a function is a sufficient condition for Lipschitz continuity and hence uniform continuity. As a result, we have the following corollary.

Corollary 1. *Suppose $g : \mathbb{R} \to \mathbb{R}$ is twice-differentiable. If g has a finite limit as $t \to \infty$ and $\ddot{g}$ is bounded, then* $\lim_{t\to\infty} \dot{g}(t) = 0$.

(d) Apply Barbalat's Lemma to V to prove a stronger statement about $e(t)$ than we could originally make with basic Lyapunov theory in part (c).

ANSWER:
For this part, we have to prove the convergence of the errors based on Barbalat's Lemma. First, we can have $\ddot{V} = -2\alpha_m e\dot{e}$. We already know that $e(t)$, $\delta_r(t)$, and $\delta_y(t)$ are bounded from part (c), we can also know that $\dot{e}(t)$ is a function of $e(t)$, δ_r, δ_y, $y(t)$, and $r(t)$.
$\dot{e} = -\alpha_m e(t)\beta\delta_r r(t) + \beta\delta_y y(t)$

We know that from problem statement, $r(t)$ is a chosen bounded reference signal. We can treat $y(t) = e(t) + y_m(t)$. Since $r(t)$ is bounded, so $y_m(t)$ is bounded. Therefore, we can know that $y(t)$ is also bounded.

Now, we are certain that $\ddot{V}$ is bounded. Then based on Corollary 1., $\lim_{t\to\infty} \dot{V}(t) = 0$. Therefore, $\lim_{t\to\infty} e(t) = 0$.

With the given control law and adaptation law, MRAC proceeds as follows. We choose a reference signal $r(t)$ to excite the reference output $y_m(t)$ and construct the input signal $u(t)$. We use $u(t)$ to excite the true model, from which the output $y(t)$ and tracking error $e(t)$ are observed. The output $y(t)$ is fed back into the control law, while the tracking error $e(t)$ is fed into the adaptation law.

(e) Apply MRAC to the unstable plant

$$\dot{y}(t) - y(t) = 3u(t).$$

That is, simulate an adaptive controller for this system that does not have access to the true model parameters $\alpha = -1$ and $\beta = 3$. The desired reference model is

$$\dot{y}_m(t) + 4y_m(t) = 4r(t),$$

with $\alpha_m = 4$ and $\beta_m = 4$. Use an adaptation gain of $\gamma = 2$, and zero initial conditions for y, y_m, k_r, and k_y. For $t \in [0, 10]$, plot both $y(t)$ and $y_m(t)$ in one figure, and $k_r(t)$, k_r^*, $k_y(t)$, and k_y^* in another figure for $r(t) \equiv 4$. Then repeat this for $r(t) = 4\sin(3t)$. Overall, you should have four figures in total. What do you notice about the trends for different reference signals? Why do you think this occurs? In your explanation, try to link your observations with the statements about $e(t)$, $\delta_r(t)$, and $\delta_y(t)$ we were able and *unable* to prove in parts (c,d).

ANSWER:
Please see the plots and code in the last few pages (appended PDF files).

For $r(t) = 4$, δ_r and δ_y didn't converge to zero. However, when using $r(t) = 4\sin(3t)$, δ_r and δ_y converged to zero. Both reference signal showed that $\lim_{t\to\infty} e(t) = 0$. This result is expected, since we have already proved that $\lim_{t\to\infty} e(t) = 0$ but not δ_r and δ_y. Therefore, δ_r and δ_y are not guaranteed to be converged to the correct values. The reason why $r(t) = 4\sin(3t)$ could make δ_r and δ_y be zero, one possible reason could be the adaption law is depending on the $r(t)$, if $\dot{k}_r(t)$ is not simply based on current $e(t)$ but it keeps changing, it can help minimize δ_r and δ_y (the true plant is controlled by $u(t)$, which also depends on $r(t)$).

Problem 3: Extremal curves

Given the functional

$$J(x) = \int_0^1 \left(\frac{1}{2}\dot{x}(t)^2 + 5x(t)\dot{x}(t) + x(t)^2 + 5x(t) \right) dt,$$

find an extremal curve $x^* : [0,1] \to \mathbb{R}$ that satisfies $x^*(0) = 1$ and $x^*(1) = 3$.

ANSWER:

We can apply CoV to this problem. The necessary condition for x^* to be an extremal, for fixed final time and fixed end point is $g_x - \frac{dg_{\dot{x}}}{dt} = 0$

$$g(x, \dot{x}, t) = \frac{1}{2}\dot{x}(t)^2 + 5x(t)\dot{x}(t) + x(t)^2 + 5x(t)$$

$$g_x = 5\dot{x} + 2x + 5$$

$$\frac{dg_{\dot{x}}}{dt} = \ddot{x} + 5\dot{x}$$

The necessary condition for x^* to be an extremal.

$$\ddot{x} - 2x = 5$$

Simply plug in $x(t) = e^{\lambda t} + c$, we will get $\lambda^2 e^{\lambda t} - 2e^{\lambda t} = 5 + 2c$. Therefore, we can know that $\lambda^2 = 2$ and $c = -\frac{5}{2}$. Therefore $x(t) = C_1 e^{\sqrt{2}t} + C_2 e^{-\sqrt{2}t} - \frac{5}{2}$.

Plug in initial and boundary conditions, we can get the following two equations.

$$x(0) = C_1 + C_2 - \frac{5}{2} = 1$$

$$x(1) = C_1 e^{\sqrt{2}} + C_2 e^{-\sqrt{2}} - \frac{5}{2} = 3$$

After solving the above two equations, we can have C_1 and C_2 to be the following two values, respectively.

$$C_1 = \frac{11 - 7e^{-\sqrt{2}}}{2(e^{\sqrt{2}} - e^{-\sqrt{2}})}$$

$$C_2 = \frac{7e^{\sqrt{2}} - 11}{2(e^{\sqrt{2}} - e^{-\sqrt{2}})}$$

$$x(t) = C_1 e^{\sqrt{2}t} + C_2 e^{-\sqrt{2}t} - \frac{5}{2}$$

Problem 4: Dubins car

The kinematics of the Dubins car are described by

$$\begin{aligned}\dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= u\end{aligned} ,$$

where $(x, y) \in \mathbb{R}^2$ is the car's position, $\theta \in \mathbb{R}$ is the car's heading, $v > 0$ is the car's constant known speed, and u is the controlled turn rate. The turn rate is bounded, i.e., $u \in [-\bar{\omega}, \bar{\omega}]$, where $\bar{\omega} > 0$ is a known constant.

The car starts at $(x, y) = (0, 0)$ with a heading of $\theta = 0$ at $t = 0$. We want the car to drive to $(x, y) = (0, c)$ in the least amount of time possible, where $c > 0$ is a given constant.

(a) Use Pontryagin's minimum principle to express the optimal control input $u^*(t)$ as a function of the optimal co-state $p^*(t) := (p_x^*(t), p_y^*(t), p_\theta^*(t)) \in \mathbb{R}^3$.

Hint: You should discover that the minimum condition for $u^*(t)$ is not informative whenever $p_\theta^*(t) \equiv \bar{p}_\theta$ for a particular fixed value $\bar{p}_\theta \in \mathbb{R}$. When such a lack of information persists over a non-trivial time interval, i.e., any time interval $[t_1, t_2]$ with $t_2 > t_1 \geq 0$, this is known as a *singular arc*. To compute $u^*(t)$ in this case, use the fact that $p_\theta^*(t) \equiv \bar{p}_\theta$ is constant in time along such an arc for this particular problem.

ANSWER:

From the problem statement, we want the car to drive to destination in the least amount of time. This means we will have the following cost function and g will be 1.

$$J = \int 1dt$$

We can have Hamiltonian as the following,

$$H = g + P^\mathsf{T} f = 1 + P_x V\cos\theta + P_y V\sin\theta + P_\theta u$$

From the above Hamiltonian equation and assumed bounded controls $u \in [-\bar{\omega}, \bar{\omega}]$, the necessary optimality conditions are

$$\frac{\partial H}{\partial P_x} = V\cos\theta$$

$$\frac{\partial H}{\partial P_y} = V\sin\theta$$

$$\frac{\partial H}{\partial P_\theta} = u$$

The above are dynamics or kinematics equation of the system.

$$\dot{P}_x^* = -\frac{\partial H}{\partial x} = 0$$

$$\dot{P}_y^* = -\frac{\partial H}{\partial y} = 0$$

$$\dot{P}_\theta^* = -\frac{\partial H}{\partial \theta} = P_x^* V \sin\theta^* - P_y^* V \cos\theta^*$$

The last is the inequality,

$$P_\theta^* u^* \leq P_\theta^* u$$

Since $u^* = \arg\min_{u \in [-\bar{\omega}, \bar{\omega}]} H$. We can simply know that from Hamiltonian, $u^* = -\bar{\omega}$ for $P_\theta > 0$ and $u^* = \bar{\omega}$ for $P_\theta < 0$. However, we have no information about how the control input u will be when $P_\theta = 0$.

If $P_\theta^*(t) \equiv \bar{P}_\theta$, then $\dot{P}_\theta^*(t) = 0$. We can get θ needs to be constant when this happens via $\dot{P}_\theta^*(t) = P_x^* V \sin\theta^* - P_y^* V \cos\theta^* = 0$, which can give us $\theta^* = \tan^{-1}\frac{P_y^*}{P_x^*}$, this means control input u has to be 0 for this case.

Therefore,
$u^* = -\bar{\omega}$ for $P_\theta > 0$
$u^* = \bar{\omega}$ for $P_\theta < 0$
$u^* = 0$ for $P_\theta = \bar{P}_\theta$

(b) Argue why $p^*(t)$ might end in a singular arc depending on the boundary conditions. Suppose we know $p^*(t)$ begins on a non-singular arc, then switches once to and ends on a singular arc. For this particular case, argue why $u^*(0) = \bar{\omega}$ and describe the optimal state trajectory $(x^*(t), y^*(t), \theta^*(t))$ and control trajectory $u^*(t)$ in words without explicitly deriving them.

ANSWER:
The problem requires the car starts at $(x, y) = (0, 0)$ with a heading of $\theta = 0$ at $t = 0$ and we want the car to drive to $(x, y) = (0, c)$ in the least amount of time possible, where $c > 0$ is a given constant. Thus, θ_f is free.

Singular arc means the control input u is 0, which means no turning rate for the car, which also means a straight line trajectory. Thus, any path starts at $(x, y) = (0, 0)$ and $\theta = 0$ with a turning rate, control input $u^*(0) = \bar{\omega}$, then becomes straight line till the end of the path, $(x, y) = (0, c)$, will make the co-state end in a singular arc. Basically, the path has an arc in the beginning plus a straight line along till the end will end in singular condition.

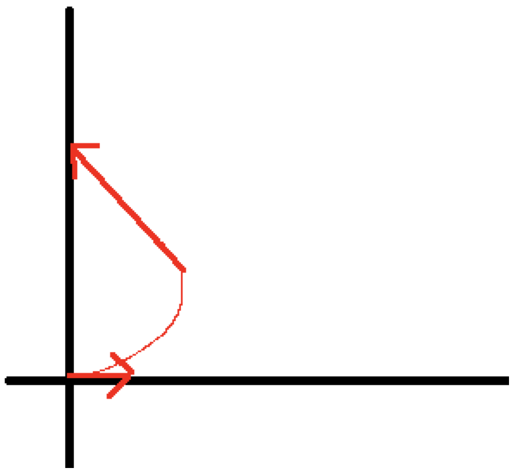


Figure 1: example of singular arc case

Problem 5: Single shooting for a unicycle

Consider the kinematic model of a unicycle

$$\begin{aligned}\dot{x}(t) &= v(t)\cos(\theta(t)),\\ \dot{y}(t) &= v(t)\sin(\theta(t)),\\ \dot{\theta}(t) &= \omega(t).\end{aligned}$$

with state $\mathbf{x} = (x, y, \theta)$ and control $\mathbf{u} = (v, \omega)$. Suppose the objective is to drive from a starting configuration to a target configuration with minimum time and control effort; specifically we want to minimize the functional

$$J = \int_0^{t_f} \left(\lambda + v(t)^2 + \omega(t)^2\right) dt,$$

where $\lambda > 0$ is a weighting factor and t_f is free, subject to the initial and terminal conditions

$$\begin{bmatrix} x(0) \\ y(0) \\ \theta(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \pi/2 \end{bmatrix}, \quad \begin{bmatrix} x(t_f) \\ y(t_f) \\ \theta(t_f) \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \\ \pi/2 \end{bmatrix}.$$

(a) Derive the Hamiltonian and necessary conditions for optimality, specifically (i) the ODE the state and costate must satisfy, (ii) the optimal control as a function of the state and costate, and (iii) the boundary conditions including the relevant transversality condition for free t_f.

ANSWER:

Hamiltonian:

$$H = \lambda + V^2 + \omega^2 + P_x V\cos\theta + P_y V\sin\theta + P_\theta \omega$$

The necessary optimality conditions:

$$\dot{x} = \frac{\partial H}{\partial P_x} = V\cos\theta$$

$$\dot{y} = \frac{\partial H}{\partial P_y} = V \sin\theta$$

$$\dot{y} = \frac{\partial H}{\partial P_\theta} = \omega$$

$$\dot{P}_x = -\frac{\partial H}{\partial x} = 0$$

$$\dot{P}_y = -\frac{\partial H}{\partial y} = 0$$

$$\dot{P}_\theta = -\frac{\partial H}{\partial \theta} = P_x V \sin\theta - P_y V \cos\theta$$

Optimal Controls (unbounded controls):

$$\frac{\partial H}{\partial u} = 0$$

$$\frac{\partial H}{\partial v} = 2v + P_x \cos\theta + P_y \sin\theta$$

$$v^* = -\frac{P_x \cos\theta}{2} - \frac{P_y \sin\theta}{2}$$

$$\frac{\partial H}{\partial \omega} = 2\omega + P_\theta$$

$$\omega^* = -\frac{P_\theta}{2}$$

Initial and Boundary Conditions:

$$x(0) = 0$$

$$y(0) = 0$$

$$\theta(0) = \frac{\pi}{2}$$

$$x(t_f) = 5$$

$$x(t_f) = 5$$

$$x(t_f) = \frac{\pi}{2}$$

$$H + \frac{\partial h}{\partial t} = H = 0$$

In practice, you might use a boundary value problem solver (e.g., `scipy.integrate.solve_bvp`), but in this problem we'll use a bit of nonlinear optimization theory to write our own!

(a) In the file `p5_unicycle_single_shooting.py`, complete the implementations of `dynamics`, `hamiltonian`, `optimal_control`, and `shooting_ode`.

ANSWER:
Please see the CODE in the last few pages (appended PDF file).

Recall from class that the simplest shooting method[1] basically boils down to guessing the correct initial costate and propagating the dynamics and costate equations until the appropriate final time (also a "guess" in free-final-time problems) at which the terminal boundary conditions are satisfied.

(a) Use the ODE integration performed by `state_and_costate_trajectories` to implement `shooting_residual`, a measure of how far off each of your terminal boundary conditions is from satisfaction, given guesses for the initial costate and final time.

ANSWER:
Please see the CODE in the last few pages (appended PDF file).

(b) Finally, in `newton_step` and `shooting_method`, implement Newton's method for finding roots of `shooting_residual`. Now, if you provide an appropriate guess for the initial costate and final time, you can run `python3 p5_unicycle_single_shooting.py` and see a plot of the optimal solution. You may find that whether or not your BVP solver converges to a solution is highly dependent on the quality of your initial guess – indeed, initialization is a major challenge when applying indirect methods for optimal control!

Hint: Recall that for finding zeros of a function $f : \mathbb{R}^n \to \mathbb{R}^n$, each iteration of Newton's method entails improving a current best guess x_k using the formula

$$x_{k+1} = x_k - \nabla f(x_k)^{-1} f(x_k),$$

where ∇f denotes the Jacobian of f.

ANSWER:
Please see the CODE in the last few pages (appended PDF files).

[1] Single shooting as opposed to, e.g., multiple shooting.

Learning goals for this problem set:

Problem 1: Learn how to construct stabilizing controllers for complicated systems by exploiting structure in the dynamics.

Problem 2: Explore the theoretical underpinnings of MRAC, and observe its behaviour on an example system in simulation.

Problem 3: Become familiar with the process of solving calculus of variations problems.

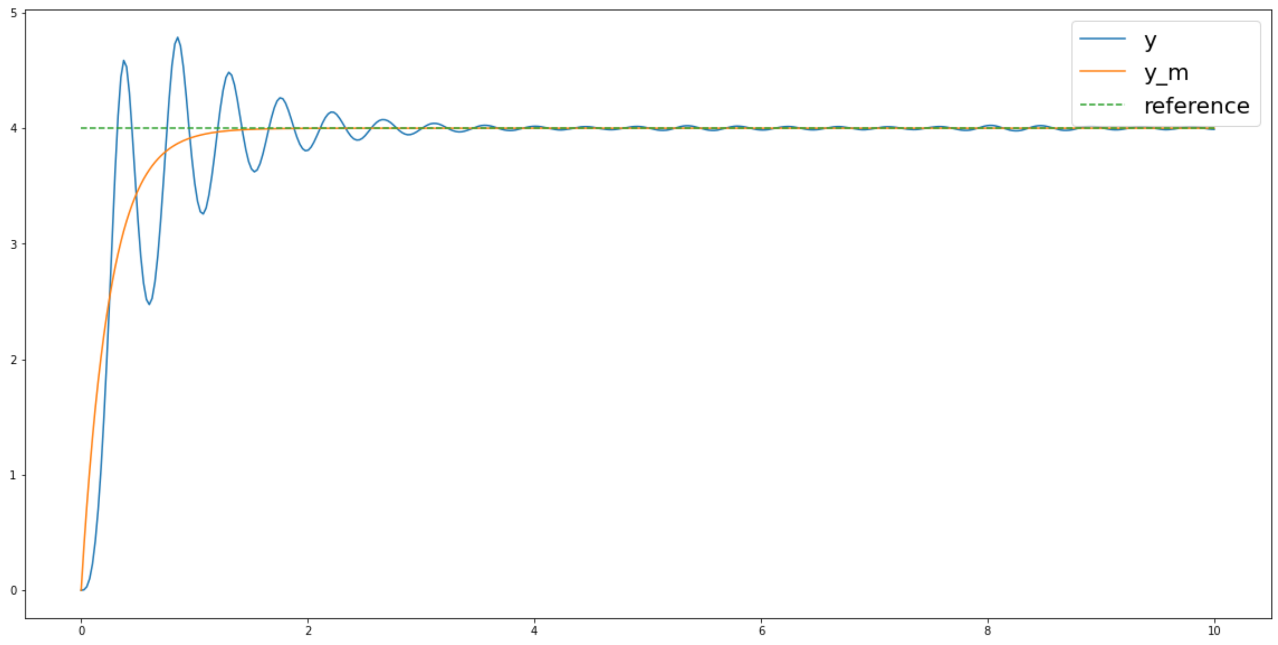
Problem 4: Learn how singular arcs can complicate analyses that use Pontryagin's minimum principle.

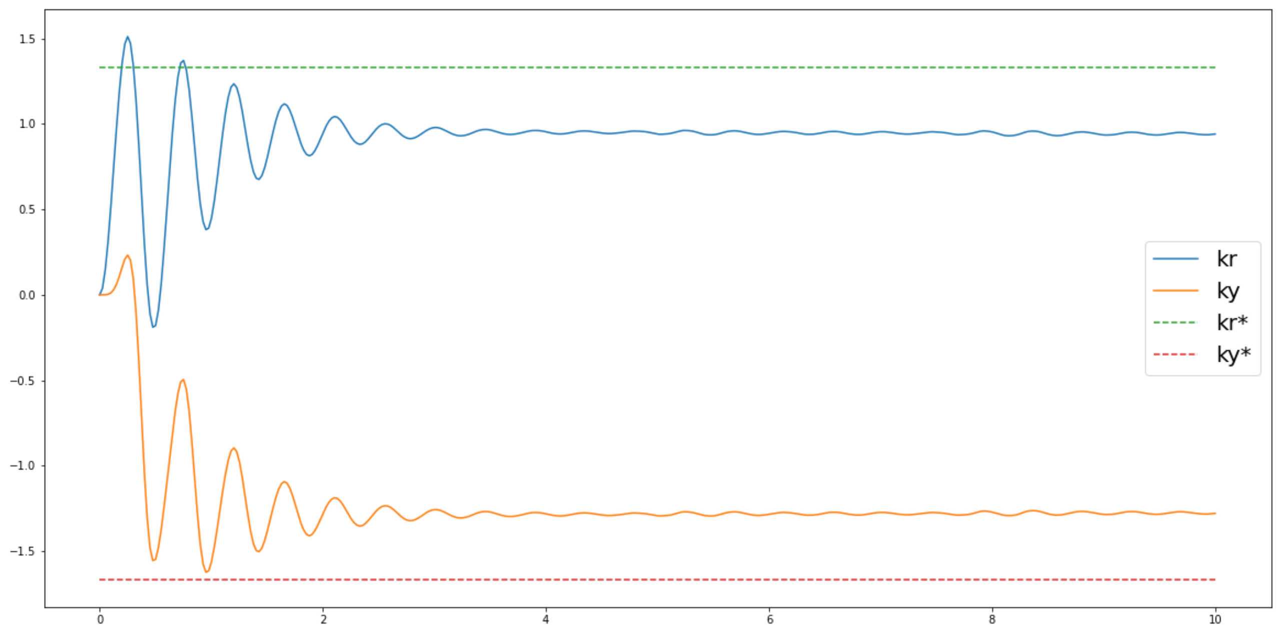
Problem 5: Implement an indirect method for optimal control and gain familiarity with JAX.

In [1]:
```python
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import solve_ivp

from ipywidgets import interact
```

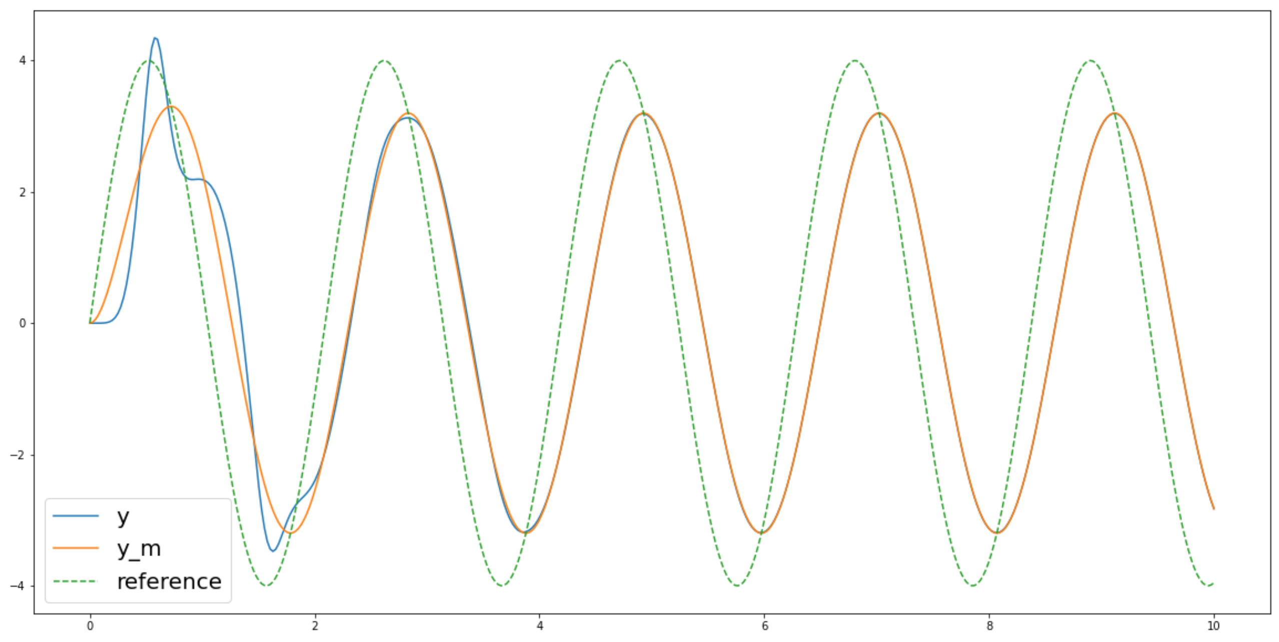
In [2]:
```python
def step(t):
    return 4*np.ones_like(t)


def sinusoidal(t):
    return 4*np.sin(3*t)


def simulate_mrac(reference_signal=step, time_horizon=10.):
    # Ideal response gains.
    kr_star = 4/3
    ky_star = -5/3

    # Adaptation gains.
    γ = 2.

    def simulation_ode(t, state_refstate_controlgains):
        y, y_m, kr, ky = state_refstate_controlgains
        r = reference_signal(t)
        u = kr*r + ky*y
        e = y - y_m
        dy_m = 4*r - 4*y_m
        dy = 3*u + y
        dkr = -γ*e*r
        dky = -γ*e*y
        return np.array([dy, dy_m, dkr, dky])

    sol = solve_ivp(simulation_ode, (0., time_horizon),
                    np.array([0., 0., 0., 0.]),
                    t_eval=np.linspace(0, time_horizon, 400))

    plt.figure(figsize=(20, 10))
    y, y_m, kr, ky = sol.y
    plt.plot(sol.t, y, label = 'y')
    plt.plot(sol.t, y_m, label = 'y_m')
    plt.plot(sol.t, reference_signal(sol.t), "--", label="reference")
    plt.legend(fontsize=20)

    plt.figure(figsize=(20,10))
    plt.plot(sol.t, kr, label = 'kr')
    plt.plot(sol.t, ky, label = 'ky')
    plt.plot(sol.t, [kr_star]*len(sol.t), '--', label = 'kr*')
    plt.plot(sol.t, [ky_star]*len(sol.t), '--', label = 'ky*')
    plt.legend(fontsize=20)
```

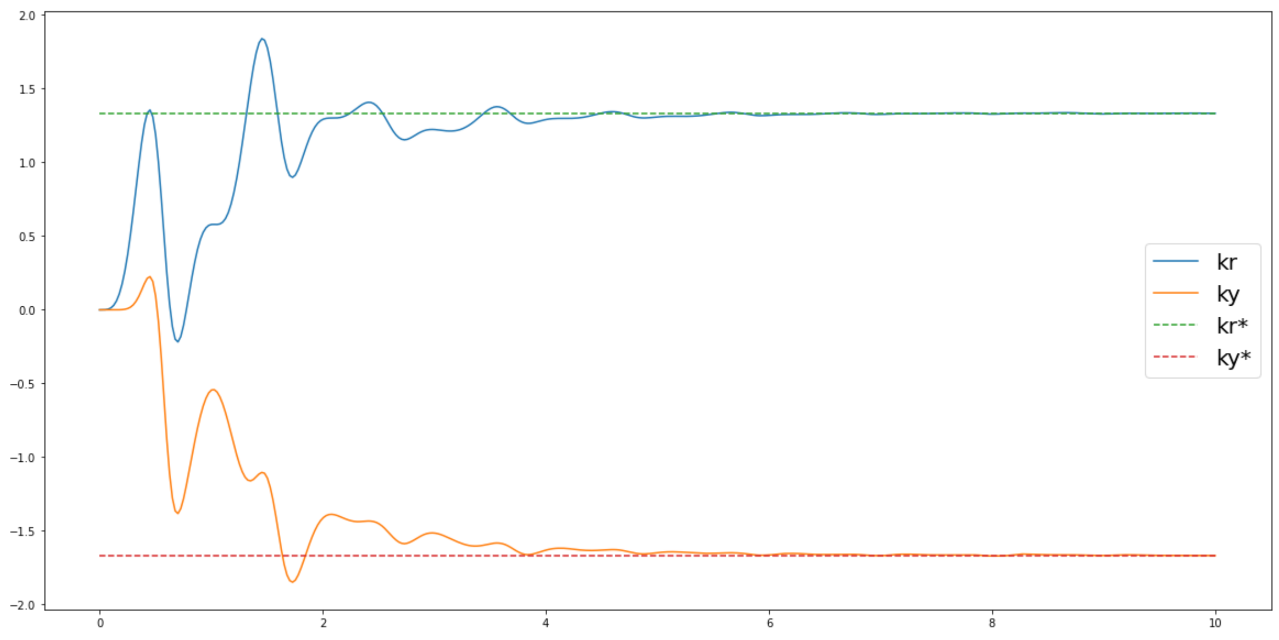
```
In [3]: simulate_mrac(reference_signal=step, time_horizon=10.)
```





```
In [4]: simulate_mrac(reference_signal=sinusoidal, time_horizon=10.)
```


y
y_m
reference
4
2
0
−2
−4
0
2
4
6
8
10

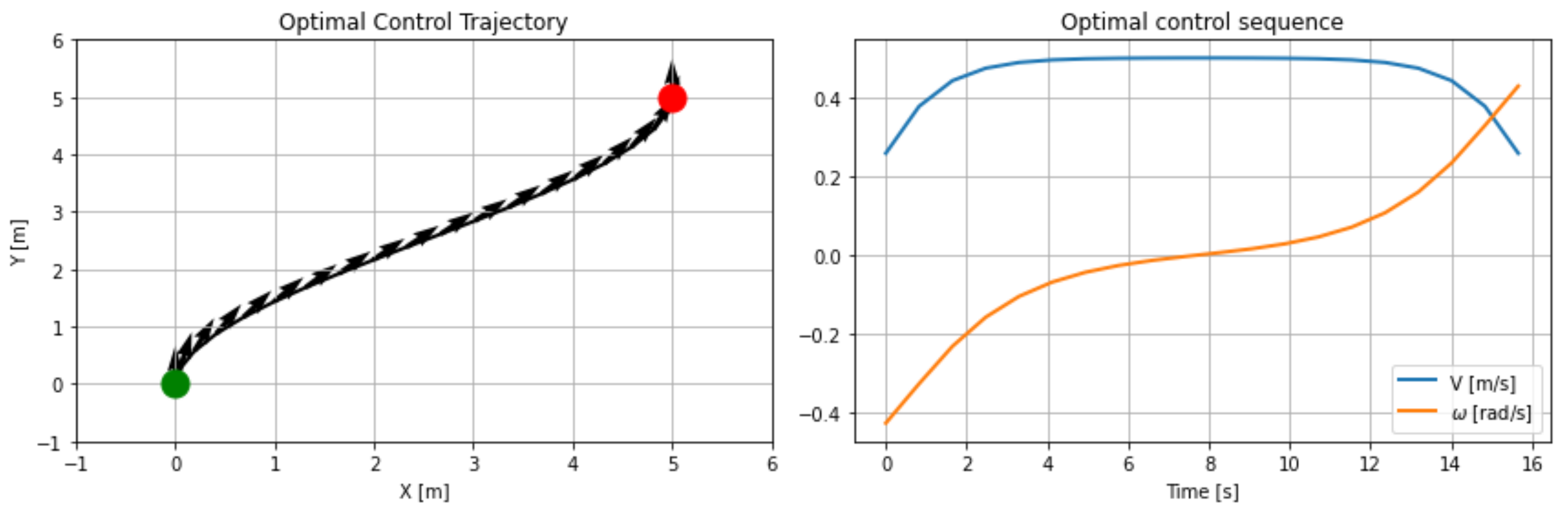

kr
ky
kr*
ky*
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
2
4
6
8
10

In [94]:
```
import jax
import jax.numpy as jnp
import matplotlib.pyplot as plt
# NOTE: It's actually best practice to use both jax.numpy (`jnp`) and regular numpy (`np`), as `np` operations are
# typically much faster than their `jnp` counterparts (at least before `jax.jit`-ing) whenever JAX-specific magic
# (automatic differentiation, vectorization, etc.) is not required. For this problem, however, since this may be your
# first experience with JAX we encourage you to use `jnp` for everything unless you know what you're doing.
# import numpy as np

from jax.experimental.ode import odeint
```

In [99]:
```
time_penalty = 0.25  # Denoted λ in the problem writeup.
initial_state = jnp.array([0., 0., jnp.pi / 2])
target_state = jnp.array([5., 5., jnp.pi / 2])


def dynamics(state, control):
    """Implements the continuous-time dynamics of a unicycle.
    Args:
        state: An array of shape (3,) containing the pose (x, y, θ) of the unicycle.
        control: An array of shape (2,) containing the linear/angular velocity controls (v, ω).
    Returns:
        The time derivative of the state. Make sure to return this as a `jnp.array`!
    """
    x, y, θ = state
    v, ω = control
    return jnp.array([v*jnp.cos(θ), v*jnp.sin(θ), ω])


def hamiltonian(state, costate, control):
    """Computes the Hamiltonian as a function of the state, costate, and control.
    Args:
        state: An array of shape (3,) containing the pose (x, y, θ) of the unicycle.
        costate: An array of shape (3,) containing the costate variables (p_x, p_y, p_θ).
        control: An array of shape (2,) containing the linear/angular velocity controls (v, ω).
    Returns:
        The scalar value of the Hamiltonian.
    """
    g = time_penalty + jnp.sum(control**2)
    f = dynamics(state, control)
    H = g + costate@f
    return H


def optimal_control(state, costate):
    """Computes the optimal control as a function of the state and costate.
    Args:
        state: An array of shape (3,) containing the pose (x, y, θ) of the unicycle.
        costate: An array of shape (3,) containing the costate variables (p_x, p_y, p_θ).
    Returns:
        An array of shape (2,) containing the optimal controls (v, ω). Make sure to return this as a `jnp.array`!
    """
    x, y, θ = state
    px, py, p_θ = costate
    v_star = -(px/2)*jnp.cos(θ)-(py/2)*jnp.sin(θ)
    ω_star = -p_θ/2
    return jnp.array([v_star,ω_star])


def shooting_ode(state_costate, t):
    """Implements the ODE that the optimal state and costate must obey.
    Args:
        state_costate: A tuple of arrays (state, costate) where
            state: An array of shape (3,) containing the pose (x, y, θ) of the unicycle.
            costate: An array of shape (3,) containing the costate variables (p_x, p_y, p_θ).
        t: Time (required for use with an ode solver; can be ignored here).
    Returns:
        A tuple of arrays (dstate_dt, dcostate_dt), the time derivatives of the state and costate.
    """
    state, costate = state_costate
    dH_dx, dH_dp = jax.grad(hamiltonian, (0, 1))(state, costate, optimal_control(state, costate))
    # HINT: There's very little left to do in this function.
    return (dH_dp, -dH_dx)


def state_and_costate_trajectories(initial_costate_and_final_time):
    """Propagates the ODE that defines the shooting method.
    Args:
        initial_costate_and_final_time: An array of shape (4,) containing (p_x(0), p_y(0), p_θ(0), t_f).
    Returns:
        A tuple of arrays (times, (states, costates)) where
            times: An array of shape (N,) containing a sequence of time points spanning [0, t_f].
            states: An array of shape (N, 3) containing the states at `times`.
            controls: An array of shape (N, 3) containing the controls at `times`.
    """
    initial_costate = initial_costate_and_final_time[:-1]
    final_time = initial_costate_and_final_time[-1]
    times = jnp.linspace(0, final_time, 20)
    return times, odeint(shooting_ode, (initial_state, initial_costate), times)


def shooting_residual(initial_costate_and_final_time):
    """Computes the residual error for the shooting method.
    Args:
        initial_costate_and_final_time: An array of shape (4,) containing (p_x(0), p_y(0), p_θ(0), t_f).
    Returns:
        The quantities we wish to drive to 0 through appropriate selection of `initial_costate_and_final_time`.
        This should be an array of shape (4,) (corresponding to 4 equations for 4 unknowns).
        Make sure to return this as a `jnp.array`!
    """
    times, (states, costates) = state_and_costate_trajectories(initial_costate_and_final_time)
    state_f = states[-1,:]
    costate_f = costates[-1,:]

    H = hamiltonian(state_f, costate_f, optimal_control(state_f, costate_f))

    return jnp.hstack((state_f-target_state,H))


# Uncomment the next line for speedier runtime (post-compilation), but a harder time debugging.
#@jax.jit
def newton_step(initial_costate_and_final_time):
    """Implements a step of Newton's method for finding zeros of `shooting_residual`.
    Args:
        initial_costate_and_final_time: An array of shape (4,) containing (p_x(0), p_y(0), p_θ(0), t_f).
    Returns:
        An improved `initial_costate_and_final_time`; the next iterate in Newton's method.
    """
    # Consider using `jnp.linalg.solve` and `jax.jacobian` to implement this as a one-liner.
    # https://jax.readthedocs.io/en/latest/_autosummary/jax.numpy.linalg.solve.html.
    # https://jax.readthedocs.io/en/latest/_autosummary/jax.jacrev.html.
    def f(x):
      return shooting_residual(x)
    jacobian = jax.jacrev(f)(initial_costate_and_final_time)
    diff = jnp.linalg.solve(jacobian, f(initial_costate_and_final_time))
    return initial_costate_and_final_time - diff


def shooting_method(initial_costate_and_final_time_guess):
    """Implements an indirect simple shooting method for solving the optimal control problem.
    Args:
        initial_costate_and_final_time_guess: An initial guess for `initial_costate_and_final_time`.
    Returns:
        An optimized `initial_costate_and_final_time`.
    """
    error = float('inf')
    eps = 1e-6
    for _ in range(1000):
      if error < eps:
        break
      next_iter = newton_step(initial_costate_and_final_time_guess)
      error = jnp.linalg.norm(next_iter-initial_costate_and_final_time_guess)
      initial_costate_and_final_time_guess = next_iter
    return initial_costate_and_final_time_guess
```

In [100…

```
initial_costate_and_final_time_guess = jnp.array([-1.,-1.,1.,20.])
times, (states, costates) = state_and_costate_trajectories(shooting_method(initial_costate_and_final_time_guess))
controls = jax.vmap(optimal_control)(states, costates)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(states[:, 0], states[:, 1], 'k-', linewidth=2)
plt.quiver(states[:, 0], states[:, 1], jnp.cos(states[:, 2]), jnp.sin(states[:, 2]))
plt.grid(True)
plt.plot(0, 0, 'go', markerfacecolor='green', markersize=15)
plt.plot(5, 5, 'ro', markerfacecolor='red', markersize=15)
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.axis([-1, 6, -1, 6])
plt.title('Optimal Control Trajectory')

plt.subplot(1, 2, 2)
plt.plot(times, controls[:, 0], linewidth=2)
plt.plot(times, controls[:, 1], linewidth=2)
plt.grid(True)
plt.xlabel('Time [s]')
plt.legend(['V [m/s]', '$\omega$ [rad/s]'], loc='best')
plt.title('Optimal control sequence')
plt.tight_layout()
plt.savefig('p5_unicycle_single_shooting.png')
plt.show()
```


Optimal Control Trajectory
X [m]
Y [m]
Optimal control sequence
Time [s]
V [m/s]
ω [rad/s]