Stanford **Spring 2023**

AA 203: Optimal and Learning-based Control
Homework #3
Due May 23 by 11:59 pm

Problem 1: Explore some theoretical underpinnings of adaptive control, and observe its behaviour on an example system in simulation.

Problem 2: Understand the basics of feasibility in MPC.

Problem 3: Introduce algorithmic details of designing terminal ingredients for MPC.

Problem 4: Apply SCP to MPC for nonlinear systems with non-convex constraints.

3.1 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

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

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

(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 $e(t) \coloneqq y(t) - y_m(t)$ and the parameter errors

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

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

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 $\dot{x} = f(t, x)$, where $x = 0$ is an equilibrium point, i.e., $f(t, 0) \equiv 0$. Suppose there exists a continuously differentiable scalar function $V(t, x)$ 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., $\|x(t)\|$ remains bounded as long as $\|x(0)\|$ is bounded.*

(c) Consider the state $x \coloneqq (e, \delta_r, \delta_y)$ 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 $\alpha_m > 0$?

Adaptive control often yields 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).

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).

3.2 MPC feasibility. Consider the discrete-time LTI system

$$x_{t+1} = Ax_t + Bu_t.$$

We want to compute a receding horizon controller for a quadratic cost function, i.e.,

$$J(x, u) = x_T^\mathsf{T} P x_T + \sum_{t=0}^{T-1} \Big(x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t \Big),$$

where $P, Q, R \succ 0$ are weight matrices. We must satisfy the state and input constraints $\|x_t\|_\infty \leq r_x$ and $\|u_t\|_\infty \leq r_u$, respectively. Also, we will enforce the terminal state constraint $\|x_T\|_\infty \leq r_T$, where we will tune $r_T \geq 0$. For $r_T = 0$, the terminal state constraint is equivalent to $x_T = 0$, while for $r_T \geq r_x$ we are just left with the original state constraint $\|x_T\|_\infty \leq r_x$.

For this problem, you will work with the starter code in `mpc_feasibility.py`. Carefully review *all* of the code in this file before you continue. Only submit code you add and any plots that are generated by the file.

(a) Implement a receding horizon controller for this system using CVXPY in the function `do_mpc`. Run the remaining code to simulate closed-loop trajectories with $r_T \geq r_x$ from two different initial states, each with either $P = I$ or P as the unique positive-definite solution to the discrete algebraic Riccati equation (DARE)

$$A^\mathsf{T} P A - P - A^\mathsf{T} P B (R + B^\mathsf{T} P B)^{-1} B^\mathsf{T} P A + Q = 0.$$

Submit your code and the plot that is generated, which displays both the realized closed-loop trajectories and the predicted open-loop trajectories at each time step. Discuss your observations of any differences between the trajectories for the different initial conditions and values of P.

(b) Finish the function `compute_roa`, which computes the region of attraction (ROA) for fixed $P \succ 0$ and different values of N and r_T. Submit your code and the plot of the different ROAs. Compare and discuss your observations of the ROAs.

Hint: While debugging your code, you can set a small `grid_dim` to reduce the amount of time it takes to compute the ROAs. However, you must submit your plot of the ROAs with at least `grid_dim = 30`.

3.3 Terminal ingredients. Consider the discrete-time LTI system $x_{t+1} = Ax_t + Bu_t$ with

$$A = \begin{bmatrix} 0.9 & 0.6 \\ 0 & 0.8 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

We want to synthesize a model predictive controller to regulate the system to the origin while minimizing the quadratic cost function

$$J(x, u) = x_T^\mathsf{T} P x_T + \sum_{t=0}^{T-1} \Big(x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t \Big),$$

subject to $\|x_t\|_2 \leq r_x$, $\|u_t\|_2 \leq r_u$, and $x_T \in \mathcal{X}_T$. For this problem, set $N = 4$, $r_x = 5$, $r_u = 1$, $Q = I$ and $R = I$.

Recall from lecture that the design of the terminal ingredients $\mathcal{X}_T$ and P is critical to guaranteeing the asymptotic stability and persistent feasibility of the resulting closed-loop system under receding horizon control.

(a) For this particular problem, explain why and how we can design $\mathcal{X}_T$ and P in an open-loop manner, i.e., by only considering the uncontrolled system $x_{t+1} = Ax_t$.

We want to find as large of a positive invariant set $\mathcal{X}_T$ for $x_{t+1} = Ax_t$ as possible that satisfies the state constraints. While maximal positive invariant sets may be computed via iterative methods using tools from computational geometry[1], we restrict our search to ellipsoids of the form

$$\mathcal{X}_T = \{x \in \mathbb{R}^n \mid x^\mathsf{T} P x \leq 1\}$$

with $P \succ 0$. Since $\text{vol}(\mathcal{X}_T) \sim \sqrt{\det(P^{-1})}$, we can formulate our search for the largest ellipsoidal $\mathcal{X}_T$ as the semi-definite program (SDP)

$$\begin{aligned} \underset{P \succ 0}{\text{maximize}} \quad & \log\det(P^{-1}) \\ \text{subject to} \quad & A^\mathsf{T} P A - P \preceq 0 \cdot \\ & I - r_x^2 P \preceq 0 \end{aligned}$$

(b) Prove that $A^\mathsf{T} P A - P \preceq 0$ and $I - r_x^2 P \preceq 0$ together are sufficient conditions for $\mathcal{X}_T$ to be a positive invariant set satisfying the state constraints.

(c) For a maximization problem, we want the objective to be a concave function. Unfortunately, the given SDP is not concave since $\log\det(P^{-1}) = -\log\det(P)$ is convex with respect to its argument $P \succ 0$. Reformulate the given SDP in P as a concave SDP in $M \coloneqq P^{-1}$.

Hint: You should use

- the fact that $A \preceq B$ if and only if $CAC \preceq CBC$ for symmetric A, B, and C where $C \succ 0$,
- the fact that $C \preceq \gamma I$ if and only if $C^{-1} \succeq \frac{1}{\gamma} I$ for $C \succ 0$ and $\gamma > 0$, and
- Schur's complement lemma, which states that

$$C - B^\mathsf{T} A^{-1} B \succeq 0 \iff \begin{bmatrix} A & B \\ B^\mathsf{T} & C \end{bmatrix} \succeq 0$$

for any conformable matrices A, B, and C where $A \succ 0$.

[1] See the MPT3 library (https://www.mpt3.org/) for some examples tailored to model predictive control.

(d) Use NumPy and CVXPY to formulate and solve the SDP for M. Plot the ellipsoids $\mathcal{X}_T$, $A\mathcal{X}_T$, and $\mathcal{X} := \{x \mid \|x\|_2^2 \leq r_x^2\}$ in the same figure. You should see that $A\mathcal{X}_T \subseteq \mathcal{X}_T \subseteq \mathcal{X}$. Submit your code and plot, and report $P := M^{-1}$ with three decimal places for each entry.

Hint: Consult the CVXPY documentation to help you write your code. Specifically, look at the list of functions in CVXPY (https://www.cvxpy.org/tutorial/functions/index.html) and the SDP example (https://www.cvxpy.org/examples/basic/sdp.html). You can write any definite constraints in the SDP with analogous semi-definite constraints (i.e., treat "$\succ$" as "$\geq$" for the purposes of writing your CVXPY code).

For plotting purposes, you can use the following Python function to generate points on the boundary of a two-dimensional ellipsoid.

```python
import numpy as np

def generate_ellipsoid_points(M, num_points=100):
    """Generate points on a 2-D ellipsoid.

    The ellipsoid is described by the equation
        `{ x | x.T @ inv(M) @ x <= 1 }`,
    where `inv(M)` denotes the inverse of the matrix argument `M`.

    The returned array has shape (num_points, 2).
    """
    L = np.linalg.cholesky(M)
    θ = np.linspace(0, 2*np.pi, num_points)
    u = np.column_stack([np.cos(θ), np.sin(θ)])
    x = u @ L.T
    return x
```

(e) Use NumPy and CVXPY to setup the MPC problem, then simulate the system with closed-loop MPC from $x_0 = (0, -4.5)$ for 15 time steps. Overlay the actual state trajectory and the planned trajectories at each time on the plot from part (d). Also, separately plot the actual control trajectory over time in a second plot. Overall, you should have two plots for this entire question. Submit both plots and all of your code.

Hint: Instead of forming the MPC problem in CVXPY during each simulation iteration, form a single CVXPY problem parameterized by the initial state and replace its value before solving (https://www.cvxpy.org/tutorial/intro/index.html#parameters).

3.4 Obstacle avoidance. In this problem, you will implement a nonlinear MPC controller to steer towards a goal position while avoiding obstacles. Consider the discrete-time nonlinear system

$$s(t+1) = f(s(t), u(t)) = s(t) + \Delta t \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ u_1(t) - \beta \dot{x}(t)|\dot{x}(t)| \\ u_2(t) - \beta \dot{y}(t)|\dot{y}(t)| \end{pmatrix},$$

where $s(t) = (x(t), y(t), \dot{x}(t), \dot{y}(t)) \in \mathbb{R}^4$ is the state of the system with position $(x(t), y(t)) \in \mathbb{R}^2$ and velocity $(\dot{x}(t), \dot{y}(t)) \in \mathbb{R}^2$, $u(t) = (u_1(t), u_2(t)) \in \mathbb{R}^2$ is the control input, $\Delta t > 0$ is a discretization constant, and $\beta > 0$ is a drag coefficient. Overall, this nonlinear system represents an Euler-discretized, two-dimensional double-integrator system subject to drag forces.

Let $s_{\cdot|t} \coloneqq \{s_{k|t}\}_{k=0}^{N}$ and $u_{\cdot|t} \coloneqq \{u_{k|t}\}_{k=0}^{N-1}$ denote N-step state and control trajectories planned from some initial state $s(t)$ with $s_{0|t} = s(t)$. We want to design a closed-loop MPC controller that optimizes such trajectories using the quadratic cost function

$$J(s_{\cdot|t}, u_{\cdot|t}) = (s_{N|t} - s_{\text{goal}})^\mathsf{T} P (s_{N|t} - s_{\text{goal}}) + \sum_{k=0}^{N-1} \Big((s_{k|t} - s_{\text{goal}})^\mathsf{T} Q (s_{k|t} - s_{\text{goal}}) + u_{k|t}^\mathsf{T} R u_{k|t} \Big),$$

with cost matrices $P \succ 0$, $Q \succ 0$, and $R \succ 0$ and goal state $s_{\text{goal}} \in \mathbb{R}^4$. We generally choose P to have much larger entries than those in Q and R, such that system state is driven towards s_{goal}.

While steering towards the goal state, we want to avoid N_{obs} obstacles. In this problem, the j^{th} obstacle is represented as a circle with center $(x_j, y_j) \in \mathbb{R}^2$ and radius $r_j > 0$. If we define a vector function $d : \mathbb{R}^n \to \mathbb{R}^{N_{\text{obs}}}$ such that the j^{th} entry is given by the *signed distance function*

$$d_j(s) = r_j - \|(x, y) - (x_j, y_j)\|_2 = r_j - \sqrt{(x - x_j)^2 + (y - y_j)^2},$$

then enforcing $d(s) \succeq 0$ would ensure the system does not collide[2] with any of the obstacles.

Overall, the MPC problem we would like to solve at each time t in closed-loop is

$$\begin{aligned} \underset{s_{\cdot|t}, u_{\cdot|t}}{\text{minimize}} \quad & (s_{N|t} - s_{\text{goal}})^\mathsf{T} P (s_{N|t} - s_{\text{goal}}) + \sum_{k=0}^{N-1} \Big((s_{k|t} - s_{\text{goal}})^\mathsf{T} Q (s_{k|t} - s_{\text{goal}}) + u_{k|t}^\mathsf{T} R u_{k|t} \Big) \\ \text{subject to} \quad & s_{0|t} = s(t) \\ & s_{k+1|t} = f(s_{k|t}, u_{k|t}), \quad \forall k \in \{0, 1, \ldots, N-1\} \\ & d(s_{k|t}) \succeq 0, \qquad\quad \forall k \in \{0, 1, \ldots, N-1\} \end{aligned}$$

There are two sources of non-convexity in this problem.

- The dynamics are nonlinear, so the equality constraint $s_{k+1|t} = f(s_{k|t}, u_{k|t})$ is non-convex.
- Each entry of the signed distance function d is convex, i.e., $-d$ is concave. Thus, $d(s_{k|t}) \succeq 0$ or equivalently $-d(s_{k|t}) \preceq 0$ is a non-convex inequality constraint.

To solve this problem, you will use *sequential convex programming (SCP)*, where for the i^{th} subproblem you will affinize the dynamics and the signed distance function around the current estimated solution $(s_{\cdot|t}^{(i)}, u_{\cdot|t}^{(i)})$. For simplicity, we will not include trust region constraints in this problem, but in practice you should always consider using them.

[2] We assume that r_j is at least slightly larger than the actual obstacle radius to avoid brushing against it.

For this problem, you will work with the starter code in `obstacle_avoidance.py`. Carefully review *all* of the code in this file before you continue. Only submit code you add to this to this starter file and any associated plots that are generated by the file.

(a) Use JAX and array broadcasting[3] to complete the function `signed_distances` with a single line of code that computes $d(s)$.

(b) Use JAX to complete the function `affinize` with two lines of code that produce an affine estimate $f(s,u) \approx As+Bu+c$ from the Taylor expansion of *any* given differentiable function f.

(c) State the convex approximation of the MPC problem around an iterate $(s^{(i)}_{\cdot|t}, u^{(i)}_{\cdot|t})$. In particular, write the constraints in terms of the affine approximations

$$\begin{aligned} f(s_{k|t}, u_{k|t}) &\approx A^{(i)}_{f,k|t} s_{k|t} + B^{(i)}_{f,k|t} u_{k|t} + c^{(i)}_{f,k|t} \\ d(s_{k|t}) &\approx A^{(i)}_{d,k|t} s_{k|t} + c^{(i)}_{d,k|t} \end{aligned},$$

and express $A^{(i)}_{f,k|t}$, $B^{(i)}_{f,k|t}$, $c^{(i)}_{f,k|t}$, $A^{(i)}_{d,k|t}$, and $c^{(i)}_{d,k|t}$ in terms of $(s^{(i)}_{\cdot|t}, u^{(i)}_{\cdot|t})$, the functions f and d, and appropriate Jacobians.

(d) The fact that we are approximating obstacle avoidance constraints may be concerning at first from a safety perspective. Prove that if $A^{(i)}_{d,k|t} s_{k|t} + c^{(i)}_{d,k|t} \succeq 0$, then $d(s_{k|t}) \succeq 0$.

Hint: A convex function $g : \mathbb{R}^n \to \mathbb{R}$ lies above any line through s that is tangent to the function g at s. For example, the quadratic function $g(s) = \|s\|_2^2$ lies entirely above the line $s = 0$, which is tangent to the function g at $s = 0$.

(e) Complete the function `scp_iteration`. Specifically, given the current iterate $(s^{(i)}_{\cdot|t}, u^{(i)}_{\cdot|t})$, use CVXPY to specify and solve the convex optimization problem you derived previously to obtain an updated solution $(s^{(i+1)}_{\cdot|t}, u^{(i+1)}_{\cdot|t})$.

(f) Complete the simulation towards the end of `obstacle_avoidance.py` to use the MPC solution from `solve_obstacle_avoidance_scp` at each time t in a closed-loop fashion.

(g) The variables `N` and `N_scp` set the MPC horizon N and the maximum number of SCP iterations N_{SCP} used to solve the MPC problem at each time, respectively. Run your completed version of `obstacle_avoidance.py` for

- $N = 5$ and $N_{\text{SCP}} = 5$,
- $N = 2$ and $N_{\text{SCP}} = 5$,
- $N = 5$ and $N_{\text{SCP}} = 2$, and
- $N = 15$ and $N_{\text{SCP}} = 2$.

The system should get closer to the goal while avoiding the obstacles. For each case, submit the generated state and control trajectory plots, and report the total computation time (`total_time`) and the total control effort (`total_control_cost`) to two decimal places. Discuss what you observe for each case.

[3] See the NumPy documention (https://numpy.org/doc/stable/user/basics.broadcasting.html) for details on array broadcasting. However, make sure to use `jax.numpy` functions!