EECS 16B Designing Information Devices and Systems II
Fall 2021 UC Berkeley

Homework 9

This homework is due on Friday, October 29, 2021, at 11:59PM. Self-grades and HW Resubmission are due on Tuesday, November 2, 2021, at 11:59PM.

1. Reading Lecture Notes

Staying up to date with lectures is an important part of the learning process in this course. Here are links to the notes that you need to read for this week: Note 11 Note 12 Note 14

(a) Why is it sufficient to check that the matrix $\mathscr{C} = \begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix}$ is full rank (i.e., why is it sufficient to check whether the system is controllable in n timesteps)?

(b) **Give a brief outline for how you would compute the Schur decomposition (i.e. upper-triangularization) of some general matrix** under the assumption that all eigenvalues are real.

(c) **What happens when we upper-triangularize a symmetric matrix?**

2. Discrete systems and their orbits

The concept of controllability exists to tell us whether or not a system can be eventually driven from any initial condition to any desired state given no disturbances, perfect knowledge of the system, and knowledge of the initial state.

For a discrete-time system with n-dimensional state $\vec{x}$ driven by a scalar input $u(t)$:

$$\vec{x}(t+1) = A\vec{x}(t) + \vec{b}u(t) \tag{1}$$

this can be checked by seeing whether the controllability matrix

$$C = \begin{bmatrix} \vec{b} & A\vec{b} & A^2\vec{b} & \cdots & A^{n-1}\vec{b} \end{bmatrix} \tag{2}$$

has a range (span of the columns) that encompasses all of $\mathbb{R}^n$. In other words, $\text{Span}(C) = \mathbb{R}^n$. If it does span the whole space, then the system (1) is controllable. If it does not, then the system is not controllable.

Everything seems to hinge on the "orbit" that the vector $\vec{b}$ takes as it repeatedly encounters A. Does this orbit confine itself to a subspace, or does it explore the whole space? (Formally, this orbit is the infinite sequence $\vec{b}, A\vec{b}, A^2\vec{b}, \ldots$)

(a) Suppose that we choose $\vec{b} = \alpha\vec{v}$ for some eigenvector $\vec{v}$ of A. That is, $A\vec{v} = \lambda\vec{v}$. **Show that the rank of C will be** 1 (i.e., show that the subspace spanned by the orbit of $\vec{b}$ through A will be one dimensional).

(b) If $\vec{b} = \vec{v}$ is an eigenvector as in the previous part, **would the system be controllable if $n > 1$?**

(c) Suppose the matrix A has at least two distinct eigenvalues $\lambda_1 \neq \lambda_2$ with corresponding eigenvectors $\vec{v}_1$ and $\vec{v}_2$. If $\vec{b} = \alpha_1\vec{v}_1 + \alpha_2\vec{v}_2$, then **show that $A^2\vec{b} = \beta_1 A\vec{b} + \beta_0\vec{b}$ for some choice of β_1 and β_0.**

(d) In the previous part, if both the $\alpha_i \neq 0$, **do the coefficients β_i depend on the exact nonzero values of the α_i?**

(e) Consequently, in the previous part, if $n > 2$, **would the system be controllable if the $\vec{b}$ was a linear combination of only two eigenvectors?**

(f) Now consider a general square matrix A (not necessarily with distinct eigenvectors, etc.) with a specific $\vec{b}$ such that the system defined by the pair $(A, \vec{b})$ is controllable. This means that the controllability matrix $[\vec{b}, A\vec{b}, \ldots, A^{n-1}\vec{b}]$ is invertible and hence gives us a basis for n-dimensional space. Consequently, for this specific vector $\vec{b}$, we know $A^n\vec{b}$ can be written in this basis. This means there exists $\{\beta_i\}_{i=0}^{n-1}$ so that:

$$A^n\vec{b} = \sum_{i=0}^{n-1} \beta_i A^i\vec{b} \tag{3}$$

Show that for all $j > 0$, the vector $\vec{w}_j = A^j\vec{b}$, also satisfies $A^n\vec{w}_j = \sum_{i=0}^{n-1} \beta_i A^i\vec{w}_j$.
(HINT: Multiply both sides of an equation by something and substitute.)

(g) Suppose your general square matrix A above has a specific $\vec{b}$ such that the system defined by the pair $\left(A, \vec{b}\right)$ is controllable. This means that it satisfies (3) above. **Use the previous part to show that in fact, $A^n - \beta_{n-1}A^{n-1} - \beta_{n-2}A^{n-2} - \ldots - \beta_1 A - \beta_0 I$ is the matrix of all zeros.**
(HINT: The previous part might provide you with a very convenient basis to use. Then remember that the signature of the zero matrix is that no matter what it multiplies, it returns zero. It a matrix takes every basis vector to zero, then it must be the zero matrix.)

It turns out that this specific polynomial $\left(\lambda^n - \beta_{n-1}\lambda^{n-1} - \ldots - \beta_1\lambda - \beta_0\right)$ must in fact be the characteristic polynomial $\left(\det\left(\lambda I - A\right)\right)$ of the matrix A. So, this argument above shows that these kinds of matrices must satisfy their own characteristic polynomials. It is also easy to see this for diagonalizable matrices A (i.e., those with an eigenbasis), but this example shows that it holds more generally for controllable matrices.

This argument can actually be used to extend to all square matrices, even those that don't have a full complement of linearly independent eigenvectors and aren't controllable with a single scalar input. When extended all the way, it is called the Cayley-Hamilton theorem. It shows that a square matrix satisfies its own characteristic polynomial — no matter what. This is an important theorem that is proved in the upper-division linear-algebra course Math 110.

It turns out that the previous parts actually help us conceptually unlock a proof that every real or complex matrix A must have at least one (possibly complex) eigenvalue/eigenvector pair. In the next couple of parts, you will do this.

(h) The first thing we're going to do is to eliminate the assumption that we have a $\vec{b}$ so that the pair $(A, \vec{b})$ is controllable.

Show that for any nonzero vector $\vec{b}$ and any n-dimensional matrix A, it must be that there exists an $\ell > 0$ and $\{\gamma_i\}_{i=0}^{\ell-1}$ so that:

$$A^\ell \vec{b} = \sum_{i=0}^{\ell-1} \gamma_i A^i \vec{b}. \tag{4}$$

(HINT: If $\vec{b}$ were such that $(A, \vec{b})$ were controllable, then you'd be done since you could invoke eq. (3). But what do you know happens when $(A, \vec{b})$ is not controllable?)

(i) Notice that we can define the polynomial $p(x) = x^\ell - \sum_{i=0}^{\ell-1} \gamma_i x^i$ using the $\{\gamma_i\}$ from eq. (4) and then we can equivalently write eq. (4) as:

$$p(A)\vec{b} = \vec{0}. \tag{5}$$

Here, we interpret $p(A)$ treating the definition of $p(x)$ like a macro and just substituting in the matrix A wherever we had the variable x in the definition. Here, we treat A^0 or the constant term as just the identity matrix.

The fundamental theorem of algebra further tells us that since $p(x)$ is a degree ℓ polynomial with complex coefficients, it can be completely factored. (This is true because the fundamental theorem of algebra guarantees us that any nonconstant polynomial over the complex numbers has at least one root — so we can factor out a term that corresponds to that root and get a polynomial of degree one lower.) This means that there exists a list $[\lambda_1, \lambda_2, \ldots, \lambda_\ell]$ (not necessarily all distinct) of possibly complex numbers so that

$$p(x) = \prod_{i=1}^{\ell} (x - \lambda_i). \tag{6}$$

This means that we also write

$$\vec{0} = p(A)\vec{b} = \left(\prod_{i=1}^{\ell} (A - \lambda_i I)\right) \vec{b}. \tag{7}$$

Show that this factorization implies that the matrix A must have an eigenvector $\vec{v}$ so that $A\vec{v} = \lambda_i\vec{v}$ for at least one of the λ_i.

(HINT: Write eq. (7) in the expanded form:

$$(A - \lambda_1 I)(A - \lambda_2 I)\cdots(A - \lambda_\ell I)\vec{b} = \vec{0} \tag{8}$$

and then reason about what can happen as you do that multiplication starting from the right and moving left. Somehow you have to end up at the zero vector. How can that happen?

Can you turn this into a procedure that will return the eigenvalue, eigenvector pair?)

3. Orthonormalization

The idea of orthonormalization is that we take a list of vectors $\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n$ and get a new list of vectors $\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_n$ such that the following properties are satisfied:

- Spans are preserved: For every $1 \leq \ell \leq n$, we know that $\text{Span}\big(\{\vec{a}_1, \ldots, \vec{a}_\ell\}\big) = \text{Span}\big(\{\vec{q}_1, \ldots, \vec{q}_\ell\}\big)$.
- The inner products of the $\vec{q}_i$ with each other are zero — they are orthogonal. That is: if $i \neq j$, we know $\vec{q}_i^{\top} \vec{q}_j = 0$.
- The $\vec{q}_i$ have unit norm whenever they are nonzero. That is, if $\vec{q}_i \neq \vec{0}$, then $\vec{q}_i^{\top} \vec{q}_i = \|\vec{q}_i\|^2 = 1$.

An algorithm for doing this was derived naturally in lecture building on what you learned in 16A about the nature of projections. This problem is about making sure that you understand it within the context of mathematical induction. Mathematical induction is a basic proof technique that is critical to understand as we build mathematical maturity. Our follow-on course CS70 assumes exposure to mathematical induction as a prerequisite and expects students to be grow to be able to craft reasonably intricate inductive proofs from scratch. Here in 16B, our goal is simply for you to be able to follow through with an induction that we set up for you, and to follow inductive arguments.

Anyway, first, let us explicitly state the iterative algorithm:

for $i = 1$ up to n **do** ▷ Iterate through the vectors
  $\vec{r}_i = \vec{a}_i - \sum_{j<i} \vec{q}_j \left(\vec{q}_j^{\top} \vec{a}_i\right)$ ▷ Find the amount of $\vec{a}_i$ that remains after we project
  if $\vec{r}_i = \vec{0}$ **then**
    $\vec{q}_i = \vec{0}$
  else
    $\vec{q}_i = \frac{\vec{r}_i}{\|\vec{r}_i\|}$ ▷ Normalize the vector.
  end if
end for

(a) From the If/Then/Else statement in the algorithm above, the third desired property holds by construction, at least for the case that $\vec{q}_i = \vec{0}$. **Show that** $\|\vec{q}_i\| = 1$ **if** $\vec{q}_i \neq \vec{0}$ (i.e., why does the "normalize the vector" line actually result in something whose norm is 1?).

(b) To establish that the spans are the same, we need to proceed by induction over ℓ. This is a classic proof by induction. (You should always strongly suspect an inductive proof lurking when you see a for loop or a recursive construction in an algorithm.)

The statement is true in the base case of $\ell = 1$ since the $\vec{q}_1$ is just a scaled version of $\vec{a}_1$. Now assume that it is true for $\ell = k - 1$. What is true? We need to translate the spans being the same into math. Namely that whenever we have an $\vec{\alpha}$ so that $\vec{y} = \sum_{j=1}^{k-1} \alpha_j \vec{a}_j$, we know there exists $\vec{\beta}$ such that $\vec{y} = \sum_{j=1}^{k-1} \beta_j \vec{q}_j$. And vice-versa: from $\vec{\beta}$ to $\vec{\alpha}$.

Show that the spans are the same for $\ell = k$ as well.

(HINT: First write out what you need to show in one direction. Then just write $\vec{a}_k$ in terms of $\vec{q}_k$ and earlier $\vec{q}_j$ and then proceed. Don't forget the case that $\vec{q}_k = \vec{0}$. Then make sure you do the reverse direction as well.)

This establishes the induction step, and since we have the base case, we know that "all dominos must fall" and the statement is true for all ℓ. This follows the "dominos" picture for induction. Establishing the inductive step shows that each domino will knock over the next domino. The base case establishes that the first domino falls. And thus, they all must fall.

(c) To establish orthogonality, we also need to do another little proof by induction over ℓ. The statement we want to prove is that for all $j < \ell$, it must be that $\vec{q}_j^{\top} \vec{q}_\ell = 0$. The base case here of $\ell = 1$ is true since there are no $j < 1$. So, we can focus on the induction part of the proof.

Here, it is convenient to use what is sometimes called "strong induction" where we assume that we know for some $k-1$ that for all $i \leq k-1$, we have that for all $j < i$, that $\vec{q}_j^{\top} \vec{q}_i = 0$ (i.e., we don't just assume that the statement is true for $\ell = k-1$, we assume it is true for all ℓ up to and including $k-1$).

(In the dominos analogy for induction, strong induction is just the fancy name for assuming that all the dominos have fallen before this one. And then showing that this one also falls. This is spiritually not that different from assuming that the previous domino has fallen and then showing that this one also falls.)

Based on this strong induction hypothesis, **show by direct calculation that for all $i \leq k$ for all $j < i$, that $\vec{q}_j^{\top} \vec{q}_i = 0$.**

(HINT: The cases $i \leq k-1$ are already covered by the induction hypothesis. So you can just focus on $i = k$. Next, notice that the case $\vec{q}_k = \vec{0}$ is also easily true. So, focus on the case $\vec{q}_k \neq \vec{0}$ and just expand what you know about $\vec{q}_k$. The strong induction hypothesis will then let you zero out a bunch of terms.)

This establishes the induction step, and since we have the base case, we know that "all dominos must fall" and the statement is true for all ℓ.

(d) It turns out that the fact that the spans are the same can be summarized in matrix form. Let $A = [\vec{a}_1, \cdots, \vec{a}_n]$ and $Q = [\vec{q}_1, \cdots, \vec{q}_n]$. If A and Q have the same column span then it must be the case that $A = QU$ where $U = [\vec{u}_1, \cdots, \vec{u}_n]$ is a square matrix. After all, this U tells us how we can find the $\vec{\beta}$ that correspond to a particular $\vec{\alpha}$ — namely $\vec{\beta} = U\vec{\alpha}$.

Show that a U can be found that is upper-triangular — that is that the i-th column $\vec{u}_i$ of U has zero entries in it for every row after the i-th position.

(HINT: Matrix multiplication tells you that $\vec{a}_i = \sum_{j=1}^{n} \vec{u}_{i_j} \vec{q}_i$. What does the algorithm tell you about this relationship? Can you figure out what $\vec{u}_{i_j}$ should be?)

Notice that this explicit construction of U can serve as part of an alternative proof of the fact that the spans are the same. The fact that the span of Q is contained within the span of A is immediate from the fact that by construction, the columns of Q are linear combinations of the columns of A. The interesting part is the other direction — that the columns of A are also all linear combinations of the columns of Q.

4. Upper Triangularization

In this problem, you need to upper-triangularize the matrix

$$A = \begin{bmatrix} 3 & -1 & 2 \\ 3 & -1 & 6 \\ -2 & 2 & -2 \end{bmatrix}$$

The eigenvalues of this matrix A are $\lambda_1 = \lambda_2 = 2$ and $\lambda_3 = -4$. We want to express A as

$$A = \begin{bmatrix} \vec{x} & \vec{y} & \vec{z} \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 & a & b \\ 0 & \lambda_2 & c \\ 0 & 0 & \lambda_3 \end{bmatrix} \cdot \begin{bmatrix} \vec{x}^\top \\ \vec{y}^\top \\ \vec{z}^\top \end{bmatrix}$$

where the $\vec{x}, \vec{y}, \vec{z}$ are orthonormal. Your goal in this problem is to compute $\vec{x}, \vec{y}, \vec{z}$ so that they satisfy the above relationship for some constants a, b, c.

Here are some potentially useful facts that we have gathered to save you some computations, you'll have to grind out the rest yourself.

$$\begin{bmatrix} 3 & -1 & 2 \\ 3 & -1 & 6 \\ -2 & 2 & -2 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ \sqrt{2} \\ 0 \end{bmatrix}.$$

We also know that

$$\left\{ \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{bmatrix}, \begin{bmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}$$

is an orthonormal basis, and

$$\begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 & -1 & 2 \\ 3 & -1 & 6 \\ -2 & 2 & -2 \end{bmatrix} \cdot \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2\sqrt{2} \\ -2\sqrt{2} & -2 \end{bmatrix}.$$

We also know that $\begin{bmatrix} 0 & -2\sqrt{2} \\ -2\sqrt{2} & -2 \end{bmatrix}$ has eigenvalues 2 and -4. The normalized eigenvector corresponding to $\lambda = 2$ is $\begin{bmatrix} -\frac{\sqrt{6}}{3} \\ \frac{\sqrt{3}}{3} \end{bmatrix}$ and $\begin{bmatrix} \frac{\sqrt{3}}{3} \\ \frac{\sqrt{6}}{3} \end{bmatrix}$ is a vector that is orthogonal to that and also has norm 1.

Based on the above information, compute $\vec{x}, \vec{y}, \vec{z}$. Show your work.

You don't have to compute the constants a, b, c in the interests of time.

5. Using upper-triangularization to solve differential equations

You know that for any square matrix A with real eigenvalues, there exists a real matrix V with orthonormal columns and a real upper triangular matrix R so that $A = VRV^\top$. In particular, to set notation explicitly:

$$V = \begin{bmatrix} \vec{v}_1, \vec{v}_2, \cdots, \vec{v}_n \end{bmatrix} \tag{9}$$

$$R = \begin{bmatrix} \vec{r}_1^\top \\ \vec{r}_2^\top \\ \vdots \\ \vec{r}_n^\top \end{bmatrix} \tag{10}$$

where the rows of the upper-triangular R look like

$$\vec{r}_1^\top = [\lambda_1, r_{1,2}, r_{1,3}, \ldots, r_{1,n}] \tag{11}$$

$$\vec{r}_2^\top = [0, \lambda_2, r_{2,3}, r_{2,4}, \ldots, r_{2,n}] \tag{12}$$

$$\vec{r}_i^\top = [\underbrace{0, \ldots, 0}_{i-1 \text{ times}}, \lambda_i, r_{i,i+1}, r_{i,i+2}, \ldots, r_{i,n}] \tag{13}$$

$$\vec{r}_n^\top = [\underbrace{0, \ldots, 0}_{n-1 \text{ times}}, \lambda_n] \tag{14}$$

where the λ_i are the eigenvalues of A.

Suppose our goal is to solve the n-dimensional system of differential equations written out in vector/matrix form as:

$$\frac{d}{dt}\vec{x}(t) = A\vec{x}(t) + \vec{u}(t), \tag{15}$$

$$\vec{x}(0) = \vec{x}_0, \tag{16}$$

where $\vec{x}_0$ is a specified initial condition and $\vec{u}(t)$ is a given vector of functions of time.

Assume that the V and R have already been computed and are accessible to you using the notation above.

Assume that you have access to a function $ScalarSolve(\lambda, y_0, \breve{u})$ that takes a real number λ, a real number y_0, and a real-valued function of time $\breve{u}$ as inputs and returns a real-valued function of time that is the solution to the scalar differential equation

$$\frac{d}{dt}y(t) = \lambda y(t) + \breve{u}(t) \tag{17}$$

with initial condition $y(0) = y_0$.

Also assume that you can do regular arithmetic using real-valued functions and it will do the right thing. So if u is a real-valued function of time, and g is also a real-valued funcion of time, then $5u + 6g$ will be a real valued function of time that evaluates to $5u(t) + 6g(t)$ at time t.

Use V, R to construct a procedure for solving this differential equation

$$\frac{d}{dt}\vec{x}(t) = A\vec{x}(t) + \vec{u}(t), \tag{18}$$

$$\vec{x}(0) = \vec{x}_0, \tag{19}$$

for $\vec{x}(t)$ by filling in the following template in the spots marked $\clubsuit, \diamondsuit, \heartsuit, \spadesuit$.

(Note: It will be useful to upper triangularize A by change of basis to get a differential equation in terms of R instead of A.)
(Note: We use the notation $\vec{v}[i]$ to be the ith component of the vector $\vec{v}$)
(HINT: The process here should be simillar to diagonalization with some modifications. Start from the last row of the system and work your way up to understand the algorithm.)

1: $\vec{\widetilde{x}}_0 = V^\top \vec{x}_0$ ▷ Change the initial condition to be in V-coordinates
2: $\vec{\widetilde{u}} = V^\top \vec{u}$ ▷ Change the external input functions to be in V-coordinates
3: **for** $i = n$ down to 1 **do** ▷ Iterate up from the bottom row
4: $\quad \breve{u}_i = \clubsuit + \sum_{j=i+1}^{n} \spadesuit$ ▷ Make the effective input for this level
5: $\quad \widetilde{x}_i = ScalarSolve(\diamondsuit, \widetilde{x}_0[i], \breve{u}_i)$ ▷ Solve this level's scalar differential equation
6: **end for**
7: $\vec{x}(t) = \heartsuit \begin{bmatrix} \widetilde{x}_1 \\ \widetilde{x}_2 \\ \vdots \\ \widetilde{x}_n \end{bmatrix} (t)$ ▷ Change back into original coordinates

(a) **Give the expression for $\heartsuit$ on line 7 of the algorithm above.** (i.e., how do you get from $\vec{\widetilde{x}}(t)$ to $\vec{x}(t)$?)

(b) **Give the expression for $\diamondsuit$ on line 5 of the algorithm above.** (i.e., what are the λ arguments to $ScalarSolve$, equation (2), for the i^{th} iteration of the for-loop?)
(HINT: Convert the differential equation to be in terms of R instead of A. It may be helpful to start with i = n and develop a general form for the ith row.)

(c) **Give the expression for $\clubsuit$ on line 4 of the algorithm above.**

(d) **Give the expression for $\spadesuit$ on line 4 of the algorithm above.**

(e) (**OPTIONAL**) Let us complete the algorithm by investigating how $ScalarSolve(\lambda, y_0, \breve{u})$ works. Consider an input that is a weighted polynomial times and exponential.

$$\breve{u}(t) = \alpha t^\beta \mathrm{e}^{\gamma t}$$

Here, α is a real constant, β is a non-negative integer, and γ is a real exponent. In addition, we will assume that $\gamma = \lambda$ for simplicity. We encourage you to attempt solving this system if $\gamma \neq \lambda$ if you are curious.

What function should $ScalarSolve(\lambda, y_0, \breve{u})$ return for the above $\breve{u}$? Remember, we are only considering the case where $\gamma = \lambda$. Express the answer in terms of α, β, γ.

6. Open-loop and closed-loop control

In the Front-End and System ID labs, we have built SIXT33N's motor control circuitry and developed a linear model for the velocity of each wheel. We are one step away from our goal: to have SIXT33N drive in a straight line! We will see how to use the model we developed in System ID to control SIXT33N's trajectory to be a straight line.

In the first part of this problem, we will see what happens if the parameters we got from system identification were not perfectly correct. In the second part, we'll see how we can understand the dynamics so that we can do closed-loop control, and then see the consequence of bounded-input bounded-output stability.

Part 1: Open-Loop Control

An open-loop controller is one in which the input is predetermined using your system model and the goal, and not adjusted at all during operation. To design an open-loop controller for your car, you would set the PWM duty-cycle value of the left and right wheels (inputs $u_L[i]$ and $u_R[i]$) such that the predicted velocity of both wheels is your target wheel velocity (v_t). You can calculate these inputs from the target velocity v_t and the $\theta_{L,R}$, and $\beta_{L,R}$ values you learned from data. In the previous homework problem and System ID lab, we have modeled the velocity of the left and right wheels as

$$v_L[i] = d_L[i+1] - d_L[i] = \theta_L u_L[i] - \beta_L; \tag{20}$$
$$v_R[i] = d_R[i+1] - d_R[i] = \theta_R u_R[i] - \beta_R \tag{21}$$

where $d_{L,R}[i]$ represent the distance traveled by each wheel.

(a) Do the open-loop control design that would give us $v_L[i] = v_R[i] = v_t$.Treat the model parameters β_L, β_R, θ_L, θ_R as symbolic quantities along with the targer velocity v_t.
i.e. Solve the model (eqs. (20) to (21)) for the inputs $u_L[i]$ and $u_R[i]$ that make the nominal velocities $v_L[i] = v_R[i] = v_t$.

(b) The challenge with reality is that it is not exactly as we think it is. The θ_L, θ_R parameters for example are learned from a finite amount data and so can be wrong. Let's use star superscripts to denote the hidden true values for the parameters, and plain (without star superscripts) to denote what we think the parameters are.
Suppose that there is 10% mismatch between θ_L in the model and θ_L^* in the physical system (i.e., $\frac{\theta_L^* - \theta_L}{\theta_L} = 0.1$.
If we used the control inputs chosen in the previous part (a), what would be the resulting velocity mismatch, $\frac{v_L[i] - v_t}{v_t}$, if $\theta_L = 2$, $\beta_L = -2.5$, and $v_t = 200$, but $\theta_L^* = 2.2$?

Above you calculated it for the left wheel, but there's no reason to believe that this discrepancy would magically be the same for the right wheel. What happens when the velocities of the two wheels disagree with each other? The robot car goes in a circle instead of a straight line. To actually go in a straight line, we need the distance traveled by both wheels to be the same.

To do this, we are going to have to use feedback control to be able to reject the disturbances that come from the fact that our model is wrong. But before we do that, we are going to need to set up a model that has some state in it that we can monitor and use to adjust the control inputs.

Here, it is useful to apply the philosophy that we saw in the trajectory tracking problem on the earlier homework. We should define our state as something that captures how far off we are from the desired behavior. Something that we would ideally like to have be zero.

This is what prompts us to define our state variable δ, to be the **difference in the *distance traveled*** between the left and right wheels at a given timestep:

$$\delta[i] := d_L[i] - d_R[i] \tag{22}$$

We have the following scalar discrete-time system equation . Here $w[i]$ is a (hopefully) bounded disturbance sequence and $f(u_L[i], u_R[i])$ is the control input to the system, which is a function of $u_L[i]$ and $u_R[i]$.

$$\delta[i+1] = \lambda_{OL}\delta[i] + f(u_L[i], u_R[i]) + w[i]. \tag{23}$$

(c) Consider the scalar discrete-time system in eq. (23). If we apply the open-loop control and the system is exactly as we have modeled it, **what is $\delta[i+1]$ in terms of $\delta[i$[? What is the eigenvalue λ_{OL} of the system with open-loop control? Would this discrete-time system be stable with open-loop control if it also had a disturbance?**

(*Hint: For open-loop control, we set the velocities to $v_L[i] = v_R[i] = v_t$. What happens when we substitute that into eq.* (20) *and eq.* (21) *and then apply the definition of $\delta[i]$ and $\delta[i+1]$?*)

Part 2: Closed-Loop Control

As we have seen in the open-loop case, if there is any mismatch in the model parameters, the velocity of the right wheel and left wheel would not necessarily be the same. To solve this problem, we must implement closed-loop control and use feedback in real time.

(d) **If we want the car to drive straight starting from some timestep $i_{\text{start}} > 0$, i.e., $v_L[i] = v_R[i]$ for $i \geq i_{\text{start}}$, what condition does this impose on $\delta[i]$ for $i \geq i_{\text{start}}$?**

(e) **How is the condition you found in (d) different from $\delta[i] = 0$ for $i \geq i_{\text{start}}$?** Assume that $i_{\text{start}} > 0$, and that $d_L[0] = 0, d_R[0] = 0$.

This is a subtlety that is worth noting and often requires one to adjust things in real systems so that control systems do what we actually want them to do.

(f) From here, assume that we have reset the distance travelled counters at the begining of this maneuver so that $\delta[0] = 0$. We will now implement a feedback controller by selecting two dimensionless positive coefficients, f_L and f_R, such that the closed loop system is stable with eigenvalue λ_{CL}. To implement closed-loop feedback control, we want to adjust $v_L[i]$ and $v_R[i]$ at each timestep by an amount that's proportional to $\delta[i]$. Not only do we want our wheel velocities to be close to some target velocity v_t, we also wish to drive $\delta[i]$ towards zero. This is in order to have the car drive straight along the initial direction it was pointed in when it started moving. If $\delta[i]$ is positive, the left wheel has traveled more distance than the right wheel, so relatively speaking, we can slow down the left wheel and speed up the right wheel to cancel this difference (i.e., drive it zero) in the next few timesteps. The action of such a control is captured by the following velocities.

$$v_L[i] = v_t - f_L\delta[i]; \tag{24}$$
$$v_R[i] = v_t + f_R\delta[i]. \tag{25}$$

Give expressions for $u_L[i]$ and $u_R[i]$ as a function of v_t, $\delta[i]$, f_L,f_R, and our nominal system parameters $\theta_L.\theta_R, \beta_L, \beta_R$, to achieve the nominal velocities above.

(g) Using the control inputs $u_L[i]$ and $u_R[i]$ found in part (f) and assuming that our nominal parameters are actually true, **write the closed-loop system equation for $\delta[i+1]$ as a function of $\delta[i]$. What is the closed-loop eigenvalue λ_{CL} for this system in terms of λ_{OL}, f_L, and f_R?**

(h) **What is the condition on f_L and f_R for the closed-loop system to be stable in the previous part?**

(i) Consider the following 10% mismatched mismatch between estimated model parameters θ_L, θ_R and the real model parameters θ_L^*, θ_R^*.

$$\frac{\theta_L^* - \theta_L}{\theta_L} = +0.1; \tag{26}$$

$$\frac{\theta_R^* - \theta_R}{\theta_R} = -0.1. \tag{27}$$

With the mismatched mismatches above, what is the corresponding system equation? What is the closed-loop eigenvalue λ_{CL} of the actual system?

(j) If there were no mismatch in the model parameters and there were no other source of disturbances $w[i]$, the state variable $\delta[i]$ wouldl eventually converge to 0 assuming the the system is stable, i.e., $|\lambda_{CL}| < 1$. However, with the mismatch introduced, $\delta[i]$ may not converge to 0 but to some other constant, which is called the steady state error $\delta_{SS} = \lim_{i\to\infty} \delta[i]$.

Remember, BIBO stability just promises that a bounded disturbance gives rise to a bounded output — it doesn't say that the result will be zero.

What is the steady state error δ_{SS} given 10% mismatch in θ_L and θ_R as in eqs. (26) to (27)? Assume that even with the mismatch, you have chosen f_L and f_R such that $|\lambda_{CL}| < 1$.

You should see that this is not zero, but instead depends on the target velocity v_t as well as the β_R and β_L constants. Physically, this reflects the fact that the car will go straight, but it might turn a little before starting to go straight. (In later courses like 128, you'll learn how to implicitly treat some of the steady mismatches as hidden states and cut down on such steady-state error.

7. Write Your Own Question And Provide a Thorough Solution.

Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning (from the bottom up) is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top level. We rarely ask you any homework questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself (e.g. making flashcards). But we don't want the same to be true about the highest level. As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams. Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't ever happen.

You need to write your own question and provide a thorough solution to it. The scope of your question should roughly overlap with the scope of this entire problem set. This is because we want you to exercise your understanding of this material, and not earlier material in the course. However, feel free to combine material here with earlier material, and clearly, you don't have to engage with everything all at once. A problem that just hits one aspect is also fine.

Note: One of the easiest ways to make your own problem is to modify an existing one. Ordinarily, we do not ask you to cite official course materials themselves as you solve problems. This is an exception. Because the problem making process involves creative inputs, you should be citing those here. It is a part of professionalism to give appropriate attribution.

Just FYI: Another easy way to make your own question is to create a Jupyter part for a problem that had no Jupyter part given, or to add additional Jupyter parts to an existing problem with Jupyter parts. This often helps you learn, especially in case you have a programming bent.

8. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.**

Contributors:

- Anant Sahai.
- Michael Danielczuk.
- Regina Eckert.
- Nathan Lambert.

- Kuan-Yun Lee.
- Kunmo Kim.
- Sidney Buchbinder.
- Daniel Abraham.
- Bozhi Yin.
- Kaitlyn Chan.
- Yi-Hsuan Shih.
- Vladimir Stojanovic.
- Moses Won.

