EECS 16B Designing Information Devices and Systems II
Spring 2021 UC Berkeley Homework 13

This homework is due on Friday, April 23, 2021, at 11:00PM. Self-grades and HW Resubmission are due on Tuesday, April 27, 2021, at 11:00PM.

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 15

(a) We know that a scalar function $f(x)$ can be linearly approximated around a particular point $x = x_*$ using Taylor's series expansion as follows:

$$f(x) \approx f(x_*) + \frac{df}{dx}\Big|_{x=x_*} \cdot (x - x_*)$$

What is the equivalent linear approximation of a multivariate scalar function $f(x, u)$ around a particular expansion point (x_*, u_*)?

(b) Now assume we have a vector valued function given by

$$\vec{f}(\vec{x}, \vec{u}) = \begin{bmatrix} f_1(\vec{x}, \vec{u}) \\ f_2(\vec{x}, \vec{u}) \\ \vdots \\ f_n(\vec{x}, \vec{u}) \end{bmatrix}$$

Let the state $\vec{x}$ be n dimensional, and control $\vec{u}$ be k dimensional. What is the linear approximation of the function $\vec{f}(\vec{x}, \vec{u})$ around a particular expansion point $(\vec{x}_*, \vec{u}_*)$?

2. Single-dimensional linearization

This is an exercise around linearization of a scalar system. The scalar nonlinear differential equation we have is

$$\frac{d}{dt}x(t) = \sin\big(x(t)\big) + u(t). \tag{1}$$

(a) The first thing we want to do is find equilibria (DC operating points) that this system can support. Suppose we want to investigate potential expansion points (x^*, u^*) with $u^* = 0$. **Sketch** $\sin(x^*)$ **for** $-4\pi \leq x^* \leq 4\pi$ **and intersect it with the horizontal line at** 0**.** This will show us the equilibria points, where $\sin(x^*) + u^* = 0$.

(b) **Show that all** $x(t) = x^*_m = m\pi$ **satisfy** (1) **together with** $u^* = 0$**, i.e.** $u(t) = u^* = 0\ \forall t$**.**

Let us zoom in on two choices: $x^*_{-1} = -\pi$ and $x^*_0 = 0$. Looking at the sketch we made, these seem like representative points.

(c) Linearize the system (1) around the equilibrium $(x^*_0, u^*) = (0, 0)$. **What is the resulting linearized scalar differential equation for** $x_\ell(t) = x(t) - x^*_0 = x(t) - 0$**, involving** $u_\ell(t) = u(t) - u^* = u(t) - 0$**?**
Remember we are ignoring the higher order terms during linearization so we have to account for those using some $w(t)$ that can be thought of as noise.

(d) For the linearized approximate system model that you found in the previous part, what happens if we try to discretize time to intervals of duration Δ? Assume now we use a piecewise constant control input $u_\ell(t) = u_\ell[n]$ in the time interval $t \in [n\Delta, (n+1)\Delta)$, where Δ is small relative to the ranges of controls applied, and that we sample the state x every Δ (that is, at every $t = n\Delta$, where n is an integer) as well. **Write out the resulting scalar discrete-time control system model, i.e. what is** $x_\ell[n+1]$ **in terms of** $x_\ell[n]$ **and** $u_\ell[n]$**?** This model is an approximation of what will happen if we actually applied a piecewise constant control input to the original nonlinear differential equation at the operating point (x^*, u^*). Here you can ignore the $w(t)$ term.

(e) **Is the (approximate) discrete-time system you found in the previous part stable or unstable?**

(f) Now linearize the system (1) around the equilibrium $(x^*_{-1}, u^*) = (-\pi, 0)$. **What is the resulting scalar differential equation for** $x_\ell(t) = x(t) - (-\pi)$ **involving** $u_\ell(t) = u(t) - 0$**?**
Again, don't forget to include the $w(t)$ term to capture the approximation error.

(g) For the linearized approximate system model that you found in the previous part, what happens if we try to discretize time to intervals of duration Δ? Assume now we use a piecewise constant control input $u_\ell(t) = u_\ell[n]$ in the time interval $t \in [n\Delta, (n+1)\Delta)$, where Δ is small relative to the ranges of controls applied, and that we sample the state x every Δ (that is, at every $t = n\Delta$, where n is an integer) as well. **Write out the resulting scalar discrete-time control system model, i.e. what is** $x_\ell[n+1]$ **in terms of** $x_\ell[n]$ **and** $u_\ell[n]$**?** This model is an approximation of what will happen if we actually applied a piecewise constant control input to the original nonlinear differential equation at the operating point (x^*, u^*). Here you can ignore the $w(t)$ term.

(h) **Is the (approximate) discrete-time system you found in the previous part stable or unstable?**

(i) Suppose for the two *linearized discrete-time systems* derived in parts (d) and (g), we chose to apply a feedback law

$$u_\ell[n] = -k(x_\ell[n] - x^*).$$

For what range of k **values, would the resulting linearized discrete-time systems be stable?** Your answer will depend on Δ.

(HINT: Your solution to part (d) should be

$$x_\ell[n+1] = e^{\Delta} x_\ell[n] + u_\ell[n](e^{\Delta} - 1)$$

and solution to part (g) should be

$$x_\ell[n+1] = e^{-\Delta} x_\ell[n] + u_\ell[n](1 - e^{-\Delta})$$

)

3. Linearizing for understanding amplification

Linearization isn't just something that is important for control, robotics, machine learning, and optimization — it is one of the standard tools used across different areas, including thinking about circuits.

The circuit below is a voltage amplifier, where the element inside the box is a bipolar junction transistor (BJT). You do not need to know what a BJT is to do this question.

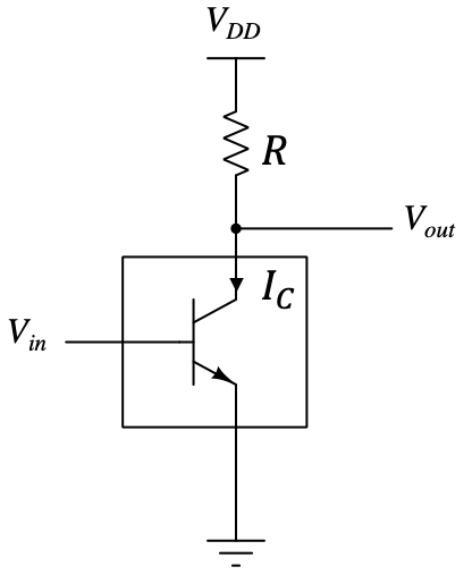


Figure 1: Voltage amplifier circuit using a BJT

The BJT in the circuit can be modeled quite accurately as a nonlinear, voltage-controlled current source, where the collector current I_C is given by:

$$I_C(V_{in}) = I_S \cdot e^{\frac{V_{in}}{V_{TH}}}, \tag{2}$$

where V_{TH} is the thermal voltage. We can assume $V_{TH} = 26\text{mV}$ at room temperature. I_S is a constant whose exact value we are not giving you because we want you to find ways of eliminating it in favor of other quantities whenever possible.

Let's consider the 2N3904 model of a BJT, where the above expression for $I_C(V_{in})$ holds as long as $0.2\text{V} < V_{out} < 40\text{V}$, and $0.1\text{mA} < I_C < 10\text{mA}$. (Note that the 2N3904 is a cheap transistor that people often use in personal projects. You can get them for 3 cents each if you buy in bulk.)

The goal of this circuit is to pick a particular point (V_{in}^*, V_{out}^*) so that any small variation δV_{in} in the input voltage V_{in} can be amplified to a relatively larger variation δV_{out} in the output voltage V_{out}. In other words, if $V_{in} = V_{in}^* + \delta V_{in}$ and $V_{out} = V_{out}^* + \delta V_{out}$, then we want the magnitude of the 'amplification gain' given by $\left|\frac{\delta V_{out}}{\delta V_{in}}\right|$ to be large. We're going to investigate this amplification using linearization.

(a) **Write a symbolic expression for V_{out} as a function of I_C, V_{DD} and R in Fig 1.**

(b) Now let's linearize I_C in the neighborhood of an input voltage V_{in}^* and a specific I_C^*. Assume that you have a found a particular pair of input voltage V_{in}^* and current I_C^* that satisfy the current equation (2).

We can look at nearby input voltages and see how much the current changes. We can write the linearized expression for the collector current around this point as:

$$I_C(V_{in}) = I_C(V_{in}^*) + m(V_{in} - V_{in}^*) = I_C^* + m\,\delta V_{in} \tag{3}$$

where $\delta V_{in} = V_{in} - V_{in}^*$ is the change in input voltage.

What is m here as a function of I_C^* and V_{TH}?

(If you take EE105, you will learn that this m is called the transconductance, which is usually written g_m, and is the single most important parameter in most analog circuit designs.)

(HINT: First just find m by taking the appropriate derivative and using the chain rule as needed. Then leverage the Taylor's series expansion of the exponential function to express it in terms of the desired quantities.)

(c) We now have a linear relationship between small changes in current and voltage, $\delta I_C = m\ \delta V_{in}$ around a known solution (I_C^*, V_{in}^*). This is called a "bias point" in circuits terminology. (This is also why related things in neural nets are called bias terms — their job is to get the nonlinearity to behave the way we want it to.)

As a reminder, **the goal of this problem is to pick a particular point (V_{in}^*, V_{out}^*) so that any small variation δV_{in} in the input voltage V_{in} can be amplified to a relatively larger variation δV_{out} in the output voltage V_{out}. In other words, if $V_{in} = V_{in}^* + \delta V_{in}$ and $V_{out} = V_{out}^* + \delta V_{out}$, then we want the magnitude of the 'amplification gain' given by $\left|\frac{\delta V_{out}}{\delta V_{in}}\right|$ to be large.**

Plug in your linearized equation for I_C in the answer from part (a). Define

$$V_{out}^* = V_{DD} - RI_C^*$$

so that it makes sense to view $V_{out} = V_{out}^* + \delta V_{out}$ when we have $V_{in} = V_{in}^* + \delta V_{in}$, and **find the approximate linear relationship between δV_{out} and δV_{in}.**

The ratio $\frac{\delta V_{out}}{\delta V_{in}}$ is called the small-signal voltage gain of this amplifier around this bias point.

(d) Assuming that $V_{DD} = 10\text{V}$, $R = 1\text{k}\Omega$, and $I_C^* = 1\text{mA}$ when $V_{in}^* = 0.65\text{V}$, **verify that the magnitude of the small-signal voltage gain $\left|\frac{\delta V_{out}}{\delta V_{in}}\right|$ between the input and the output around this bias point is approximately 38.**

(HINT: Remember $V_{TH} = 26\text{mV}$)

(e) If $I_C^* = 9\text{mA}$ when $V_{in}^* = 0.7\text{V}$ with all other parameters remaining fixed, **verify that the magnitude of the small-signal voltage gain $\left|\frac{\delta V_{out}}{\delta V_{in}}\right|$ between the input and the output around this bias point is approximately 350.**

(f) If you wished to make an amplifier with as large of a small signal gain as possible, which operating (bias) point would you choose among $V_{in}^* = 0.65\text{V}$ (part d) and $V_{in}^* = 0.7\text{V}$ (part e)?

This shows you how by appropriately biasing (choosing an operating point), we can adjust what our gain is for small signals. Although here, we just wanted to show you this as a simple application of linearization, these ideas are developed a lot further in 105, 140, and other courses to create things like op-amps and other analog information-processing systems.

4. Inverse Kinematics

Inverse Kinematics is critical in robotics, control, and computer graphics applications.

We need to be able to go backward from what we want to have happen in the real (or virtual) world to how to set parameters.

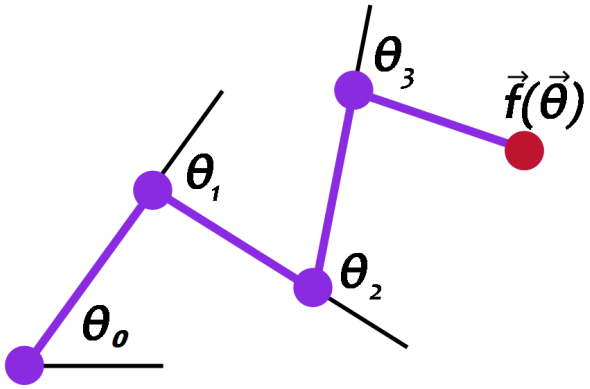


Figure 2: An example of an arm parameterized by θ_0, θ_1, θ_2, and θ_3 with the end effector at point $\vec{f}(\vec{\theta})$.

Suppose you have a robotic arm composed of several rotating joints.

The lengths r_i of the arm are fixed, but you can control the arm by specifying the amount of rotation θ_i for each joint. If we have an arm with four joints, it can be parameterized by:

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}. \tag{4}$$

Suppose further that we have some target $\vec{t} \in \mathbb{R}^2$, which represents a point in the 2D space, and we would like for the end of the arm, called the end effector, to reach for the target. We have some function $\vec{f}(\vec{\theta})$ that rotates each joint of the arm according to the input and returns the position of the end effector. Figure 2 shows a visualization of an arm rotated by $\vec{\theta}$. To make the arm reach for the target $\vec{t}$, we want to find where the function $\vec{g}$ defined as

$$\vec{\theta} \in \mathbb{R}^4 \mapsto \vec{g}(\vec{\theta}) = \vec{f}(\vec{\theta}) - \vec{t} \tag{5}$$

is equal to $\vec{0}$. To accomplish this, we use the spirit of Newton's method for solving potentially nonlinear equations. (This is related to the spirit of the earlier problem on using iterative ways of solving least-squares problems.)

You might have seen Newton's method in your calculus course in the 1-D case. In this case, you have a real function g of a single parameter θ and we want to find a $\hat{\theta}$ so that $g(\hat{\theta}) = 0$. The method is the following. Step i of Newton's method does the following:

(a) Linearize g around $\theta^{(i)}$, the current estimate of $\hat{\theta}$: $\hat{g}^{(i)}(\theta) = g(\theta^{(i)}) + g'(\theta^{(i)})(\theta - \theta^{(i)})$

(b) $\theta^{(i+1)}$ solves $\hat{g}^{(i)}(\theta) = 0$. So we have

$$0 = g(\theta^{(i)}) + g'(\theta^{(i)})(\theta - \theta^{(i)}) \tag{6}$$

$$\theta = -\frac{g(\theta^{(i)})}{g'(\theta^{(i)})} + \theta^{(i)} \tag{7}$$

Therefore $\theta^{(i+1)} = -\frac{g(\theta^{(i)})}{g'(\theta^{(i)})} + \theta^{(i)}$. Note also that doing this is solving: $g'(\theta^{(i)})(\theta - \theta^{(i)}) = g(\theta^{(i)})$, which reduces to "inverting" the linear operator $z \mapsto g'(\theta^{(i)})z$.

We will iterate this until $g(\theta)$ is close enough to 0 for our application. In practice, instead of solving exactly for $\hat{g}^{(i)} = 0$, in the second step of iteration i, we may chose to move $\theta^{(i)}$ by a fixed step-size η in the direction that the first-order approximation to the function suggests, but not all the way. This is done because the derivative $g'(\theta)$ where $\hat{g}(\theta) = 0$ might be very different from $g'(\theta^{(i+1)})$ where $\theta^{(i+1)}$. After all, linearization is only valid in a local neighborhood. (This is the spirit of gradient descent as well.)

While you might have seen Newton's method as described above in your calculus courses, you might not have seen the vector-generalization of it. It follows exactly the same spirit. The first-order approximation to the vector valued function $\vec{g}(\vec{\theta})$ at $\vec{\theta}^{(i)}$ is now $\vec{g}(\vec{\theta}^{(i)}) + J_{\vec{g}}(\vec{\theta}^{(i)})(\vec{\theta} - \vec{\theta}^{(i)})$ where $J_{\vec{g}}(\vec{\theta})$ is the Jacobian matrix of the function $\vec{g}(\vec{\theta})$. For this problem, we will be using a robotic arm with 4 joints in a 2-dimensional space. Therefore, the Jacobian of $\vec{g}(\vec{\theta})$ will be a 2x4 matrix, and it is computed by calculating the partial derivatives of $\vec{g}(\vec{\theta})$:

$$J_{\vec{g}} = \begin{bmatrix} \frac{\partial g_x(\vec{\theta})}{\partial \theta_1} & \frac{\partial g_x(\vec{\theta})}{\partial \theta_2} & \frac{\partial g_x(\vec{\theta})}{\partial \theta_3} & \frac{\partial g_x(\vec{\theta})}{\partial \theta_4} \\ \frac{\partial g_y(\vec{\theta})}{\partial \theta_1} & \frac{\partial g_y(\vec{\theta})}{\partial \theta_2} & \frac{\partial g_y(\vec{\theta})}{\partial \theta_3} & \frac{\partial g_y(\vec{\theta})}{\partial \theta_4} \end{bmatrix}. \tag{8}$$

In this notation, we use $\vec{g}(\vec{\theta}) = [g_x(\vec{\theta}) \quad g_y(\vec{\theta})]^T$ where $g_x(\vec{\theta})$ is the x coordinate of the end effector and $g_y(\vec{\theta})$ is the y coordinate in our 2D space. There is nothing mysterious about this, if you think about it, this matrix of partial derivatives (a partial derivative is just a regular derivative with respect to a particular variable, treating all the other variables as constants) is the natural n-d candidate to replace g'.

The Newton algorithm in this case is an iterative method that gives us successively better estimates for our vector $\hat{\vec{\theta}}$. If we start with some guess $\vec{\theta}^{(i)}$, then the next guess is given by

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta J_{\vec{g}}^{\dagger}(\vec{\theta}^{(i)})\vec{g}(\vec{\theta}^{(i)}) \tag{9}$$

where η is adjusted to determine how large of a step we make between $\vec{\theta}^{(i)}$ and $\vec{\theta}^{(i+1)}$. Notice that we need to invert the Jacobian matrix of first-partial-derivatives, and this matrix is not square. It is in fact a wide matrix. Fortunately, we know how to "invert" wide matrices, using the Moore Penrose pseudo-inverse that you saw in a previous homework. The minimality property of the Moore-Penrose pseudoinverse that we studied then is useful here because we would rather take small steps than big ones. And when tracking a moving reference, we'd like to have the joint angles change in a minimal way rather than in some very convoluted fashion.

The following problem will guide you step-by-step through the implementation of the pseudoinverse. The three steps of the psuedoinverse algorithm are:

- First, compute the compact-form SVD of the input matrix.
- Next, we compute Σ^{-1} by inverting each singular value σ_i.
- Finally, we compute the pseudoinverse by multiplying the matrices together in the right order.

The next three parts guide you through the code to be written for the pseudoinverse.

(a) In the "pseudoinverse" function, **compute** the SVD of the input matrix A by using the appropriate NumPy function.

(HINT: It is useful to read the documentation for the numpy functions involved so that you call them with the right arguments. For example, for this problem you need to call `svd(A, full_matrices=False)`*, What is the default? What do you want the function to return? What exactly does the SVD function return in numpy?)*

(b) To save memory space, the NumPy algorithm returns the matrix Σ as a one-dimensional array of the singular values.

Use this vector to **compute the diagonal entries of** Σ^{-1}. Be careful of numerical issues: first threshold the singular values, and only invert the singular values above a certain value ϵ, considering smaller ones are 0.

The reason is that you don't want to have very big entries in the pseudoinverse because that will defeat the point of you using a small step-size η to stay within the rough neighborhood that your linear approximation is valid. So you need to stop that from happening. That is what considering small singular values as being 0 does.

(c) We now have all of the parts to compute the relevant pseudoinverse of A. **Add this computation to the function.**

(HINT: `np.diag` *can be a very useful tool in converting a vector into a square diagonal matrix. Also remember that numpy knows how to multiply matrices and using* `.T` *compute transposes.)*

(d) There are three test cases that you can use to determine if your pseudoinverse function works correctly. In the first case, the arm is able to reach the target, and the end of the arm will be touching the target. In the second case, the arm should be pointing in a straight line towards the blue circle. The last case is the same as the second with the addition that a singular value will be very close to zero to test your pseudoinverse function's ability to handle small singular values. There is also an animated test case that will move the target in and out of the reach of the arm. Verify that the arm follows the target correctly and points towards the target when it is out of reach.

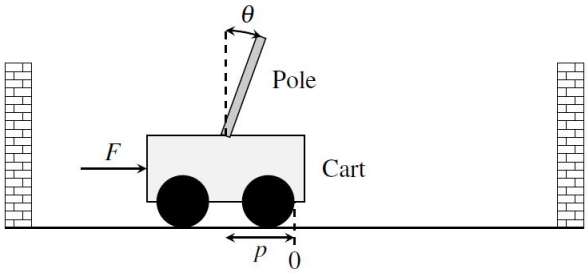
5. Segway Tours

A segway is a stand on two wheels, and can be thought of as an inverted pendulum. The segway works by applying a force (through the spinning wheels) to the base of the segway, This controls both the position on the segway and the angle of the stand. As the driver pushes on the stand, the segway tries to bring itself back to the upright position, and it can only do this by moving the base.

You may recall analyzing a problem related to segway in 16A homework. You were given a linear discrete time representation of the segway dynamics, and were asked to analyze if it's possible to make the segway reach some desired states. Now, we will see how to derive the linear discrete time system from the equations of motion and do some further detailed analysis.

Note that this problem is completely independent from the 16A version and does not require any prior knowledge or results of the previous problem.

Is it possible for the segway to be brought upright and to a stop from any initial configuration? There is only one input (force) used to control two outputs (position and angle). Let's model the segway as a cart-pole system and analyze.



A cart-pole system can be fully described by its position p, velocity $\frac{dp}{dt}$, angle θ, and angular velocity $\frac{d\theta}{dt}$. We can write this as the continuous time state vector $\vec{x}$ as follows:

$$\vec{x} = \begin{bmatrix} p \\ \frac{dp}{dt} \\ \theta \\ \frac{d\theta}{dt} \end{bmatrix}$$

The input to this system is a scalar quantity $u(t)$ at time t, which is the force F applied to the cart (or base of the segway). Let the co-efficient of friction be k.

The equations of motion for this system are as follows:

$$\begin{aligned} \frac{d^2p}{dt^2} &= \frac{1}{\frac{M}{m} + \sin^2\theta} \left(\frac{u}{m} + \left(\frac{d\theta}{dt}\right)^2 l \sin\theta - g \sin\theta \cos\theta - \frac{k}{m}\frac{dp}{dt} \right) \\ \frac{d^2\theta}{dt^2} &= \frac{1}{l\left(\frac{M}{m} + \sin^2\theta\right)} \left(-\frac{u}{m}\cos\theta - \left(\frac{d\theta}{dt}\right)^2 l \cos\theta \sin\theta + \frac{M+m}{m} g \sin\theta + \frac{k}{m}\frac{dp}{dt}\cos\theta \right) \end{aligned} \tag{10}$$

The derivation of these equations is a mechanics problem and not in 16B syllabus, but interested students can look up the details online.

(a) First let us linearize the system of equations in (10) about the upright position at rest, i.e. $\theta_* = 0$ and $\frac{d\theta}{dt}_* = 0$. **Show that the linearized system of equations is given by the following state space form:**

$$\frac{d\vec{x}(t)}{dt} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{k}{M} & -\frac{m}{M}g & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{k}{Ml} & \frac{M+m}{Ml}g & 0 \end{bmatrix}}_{A} \vec{x}(t) + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ -\frac{1}{Ml} \end{bmatrix}}_{\vec{b}} u(t) \tag{11}$$

(HINT: Since we are linearizing around $\theta_* = 0$ and $\frac{d\theta}{dt}_* = 0$, you can use the following approximations for small values of θ:

$$\begin{aligned} \sin\theta &\approx \theta \\ \sin^2\theta &\approx 0 \\ \cos\theta &\approx 1 \\ \left(\frac{d\theta}{dt}\right)^2 &\approx 0. \end{aligned}$$

You do not have to do the full linearization using Taylor series, you can just substitute the values above. You will get the same answer as doing the linear Taylor series approximation.)

(b) For all subsequent parts, assume that $m = 1$, $M = 10$, $g = 10$, $l = 1$ and $k = 0.1$. Let's consider the discrete time representation of the state space (11) at time $t = n\Delta$. For simplicity, assume $\Delta = 1$. The discrete time state $\vec{x}_d$ follows the following linear model:

$$\vec{x}_d[n+1] = A_d\vec{x}_d[n] + \vec{b}_d u_d[n] \tag{12}$$

where $A_d \in \mathbb{R}^{4\times4}$ and $\vec{b}_d \in \mathbb{R}^{4\times1}$. **Find A_d and $\vec{b}_d$.** Use the Jupyter notebook **segway.ipynb** for numerical calculations, and approximate the results to 2 or 3 significant figures.
(HINT: Recall that the continuous time scalar differential equation

$$\frac{dz(t)}{dt} = \lambda z(t) + cw(t)$$

can be represented in discrete time ($n\Delta = t$) as follows:

$$z_d[n+1] = \begin{cases} (e^{\lambda\Delta}) \cdot z_d[n] + \left(\frac{e^{\lambda\Delta}-1}{\lambda}\right) \cdot cw_d[n] & \text{if } \lambda \neq 0 \\ (1) \cdot z_d[n] + (\Delta) \cdot cw_d[n] & \text{if } \lambda = 0 \end{cases}$$

Use the eigendecompostion of $A = V\Lambda V^{-1}$ to do change of basis variables, and you should finally reach

$$\vec{x}_d[n+1] = \underbrace{V\Lambda_d V^{-1}}_{A_d}\vec{x}_d[n] + \underbrace{VM_dV^{-1}\vec{b}}_{\vec{b}_d}u_d[n]$$

What are the elements of Λ_d and M_d in terms of the elements of Λ?)

(c) **Show that the discrete time system in (12) is controllable by using the appropriate matrix in the Jupyter notebook.**
(HINT: Is the controllability matrix full rank? You have to use numerical values of A_d and $\vec{b}_d$ from the previous part. Use the Jupyter notebook for all numerical calculations.)

(d) Since the discrete time system is controllable, it is possible to reach any final state $\vec{x}_{d,f}$ starting from initial state $\vec{x}_{d,i}$ using an appropriate sequence of inputs in exactly 4 steps, provided that the deviations are small enough so that the linearization approximation is valid. **Set up a set of linear equations to solve for the $u_d[0]$, $u_d[1]$, $u_d[2]$, $u_d[3]$ given the initial and final states. Find the input sequence to reach the upright position $\vec{x}_{d,f} = \vec{x}_d[4] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ starting from an initial state $\vec{x}_{d,i} = \vec{x}_d[0] = \begin{bmatrix} -2 \\ 3.1 \\ 0.3 \\ -0.6 \end{bmatrix}$.** Use the Jupyter notebook for all numerical calculations and simulation. **Explain qualitatively what you observe from the segway simulation.**
(HINT: Use (12) and loop unrolling to express $\vec{x}_d[4]$ as a linear combination of $\vec{x}_d[0]$, $u_d[3]$, $u_d[2]$, $u_d[1]$, $u_d[0]$.)

(e) Now suppose we try to use an initial state $\vec{x}_{d,i} = \vec{x}_d[0] = \begin{bmatrix} -2 \\ 3.1 \\ 3.3 \\ -0.6 \end{bmatrix}$ that does not satisfy the linearization constraint, since $\theta_i = 3.3$ is very far from the linearization point $\theta_* = 0$. **Using the equations derived in the previous part, use the Jupyter notebook to determine the input sequence to reach the same final upright position. Explain qualitatively what you observe from the segway simulation.** Use the Jupyter notebook for all numerical calculations and simulation.

Compare the simulation results in parts (d) and (e). In both cases, the segway finally stabilizes to an upright position at rest. However, in part (d) the behavior of the segway looks more realistic whereas in part (e) it is doing some wild unexpected rotations.

This is because the linearization approximation is valid with the small initial values of θ and $\frac{d\theta}{dt}$ in part (d). So this discrete time linear model is a good representation of the original continuous time non-linear system. Hence the trajectory taken by the segway from the initial to the final position is similar to what we may expect from real life physics.

However in part (e), the linearization approximation is violated. The model still converges to the final upright position because (12) is controllable as we proved in part (c). However, since the approximation is not valid anymore, this discrete time linear model is **not** a good representation of the original continuous time non-linear system. Hence the trajectory is extremely weird with the segway undergoing a few full rotations, and does not match what we would expect from the real system.

We can still analyze the system in continuous time by directly solving the set of non-linear differential equations in (10) (out of 16B scope) or in discrete time using a discretized version of (10). Note that there are two independent distinctions we are making, i.e. continuous vs discrete, and linear vs non-linear. Part (e) failed because it's beyond the scope of the linear model, not because we are using a discrete time system. A non-linear discrete time analysis would also give the correct solution.

(f) We have attached two videos: **cart_pole_non_linearized.mp4** and **cart_pole_linear.mp4**. These videos show the behavior of the continuous time segway system before (10) and after (11) we linearize

it, when no control input is applied. More specifically, we use a starting state of

$$\vec{x}(0) = \begin{bmatrix} 0 \\ 0 \\ 0.1\text{rad} \\ 0 \end{bmatrix}$$

and we set the input $u(t) = 0$ for the entire simulation. **Contrasting the non-linear and linearized continuous time simulations in the two videos, do you notice any differences in the trajectory when the angle θ gets large? Why is this the case?**

6. 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.)

Contributors:

- Ayan Biswas.
- Anant Sahai.
- Kris Pister.
- Alex Devonport.
- Regina Eckert.
- Stephen Bailey.
- Daniel Abraham.