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

Homework 12

This homework is due on Friday, November 19, 2021, at 11:59PM. Self-grades and HW Resubmission are due on Tuesday, November 23, 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 19

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

$$f(x) \approx f(x^\star) + \frac{\mathrm{d}f}{\mathrm{d}x}(x^\star) \cdot (x - x^\star)$$

What is the equivalent linear approximation of a multivariate scalar function $f(x, u)$ around a particular expansion point $(x^\star, u^\star)$?

(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 $(\vec{x}^\star, \vec{u}^\star)$?**

2. Low Rank Approximation of a Matrix *Optional to do, but completely in scope.*

In this question we will study the so called "low rank approximation" problem. As the name implies, consider an arbitrary *wide* matrix $X \in \mathbb{R}^{m \times n}$, with $m \leq n$. [1] We are interested in finding another matrix $\widehat{X}$ having specified lower rank k, such that $\widehat{X}$ is "closest" to X, i.e.,

$$\min_{\widehat{X}} \quad \left\| X - \widehat{X} \right\|_F \tag{1}$$

$$\text{subject to} \quad \operatorname{rank}(\widehat{X}) \leq k \tag{2}$$

This problem goes to the heart of how we use the SVD for dimensionality reduction and to look at data. If we view a data matrix as a collection of columns where each of the columns is a different data point, then a rank-k approximation to that matrix is a collection of columns all of which represent points that are all on a k-dimensional subspace. *This discovery of hidden subspace structure is what finding low-rank approximations is truly about.*

(a) First, let's understand one of the simpler interpretations of why rank-r approximations to huge matrices are so useful. To specify an arbitrary $m \times n$ matrix, we have to choose mn independent elements (entries). In other words, an arbitrary $m \times n$ matrix has mn degrees of freedom. **How many independent elements (degrees of freedom) do we have to know to specify a rank r matrix of the same $m \times n$ size?**

You will see that this number is a lot less than mn, when r is small. By storing only these independent elements, we can compress our $m \times n$ matrix, reducing the amount we need to store to a small fraction of the original size. By finding the *best* low-rank approximation, even to a matrix which is high rank, we can achieve the best possible trade-off between compression efficiency and reconstruction error.

(HINT: Think about outer-product representations for a rank r matrix, for example, that given by the SVD.)

(b) To understand this problem, we will have to also think about what it might mean to approximate a matrix. For this, we use a natural notion of error for matrices.

In the previous homework, you were introduced to the Frobenius norm — which involved treating a matrix as though it was just a big long vector filled with its entries.

Similar to how the regular vector norm measures the error in a least-squares context, the Frobenius norm measures the error in this matrix approximation context.

We will now build up some properties of the Frobenius norm. More specifically, suppose we view a matrix A as a list of columns:

$$A = \begin{bmatrix} \vec{a}_1 & \vec{a}_2 & \cdots & \vec{a}_n \end{bmatrix} \tag{3}$$

Show that the Frobenius norm $\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |A_{ij}|^2}$ for a matrix A can be understood in terms of the regular Euclidean norms of its columns as:

$$\|A\|_F = \sqrt{\sum_{j=1}^{n} \left\| \vec{a}_j \right\|^2} \tag{4}$$

This is useful because it justifies using Frobenius norm as a way to measure the length of a matrix when we are really viewing the matrix as a collection of columns.

[1] It doesn't matter if X is actually wide, square, or tall; for the sake of simplicity, we consider it as wide.

(c) Now we want to get into the "hidden subspace" aspect of this problem. To do this, we need a way of talking about a potential subspace. To define a subspace, we need a basis. For convenience, we might as well think of an orthonormal basis. Let the matrix B consist of k orthonormal columns. The matrix B defines a subspace S of dimension k; in particular, $S = \text{Col}(B)$. Our underlying goal is to find an optimal such subspace S for approximating the data in X and equivalently, to find an optimal basis B. (Note here that $B \in \mathbb{R}^{m \times k}$. There are k columns, each of which is an m-dimensional vector.)

Before we worry about finding such a basis or subspace, it is good to understand how we would approximate X using it. We already know how to approximate a single vector by a bunch of vectors in a subspace – we take the projection. So to approximate a whole matrix X by vectors in a subspace, we project each of the columns of X into the subspace simultaneously. By using the least squares formula and the fact that $B^\top B = I$, we get that the result of this projection is $BB^\top X$.

To understand the quality of this projection, first **show that** $\left\|X - BB^\top X\right\|_F^2 = \|X\|_F^2 - \left\|B^\top X\right\|_F^2$**.**

(HINT: Let $\vec{x}_i$ be the i^{th} column of X. Use the Pythagorean theorem – for vectors – to show that $\left\|\vec{x}_i - BB^\top \vec{x}_i\right\|^2 = \|\vec{x}_i\|^2 - \left\|B^\top \vec{x}_i\right\|^2$. Then use this to show what we want.)

We can see from the previous part that since $\|X\|_F^2$ is outside our control, finding a matrix B with k orthonormal columns that minimizes $\left\|X - BB^\top X\right\|_F^2$ is the same as finding a matrix B with k orthonormal columns that maximizes $\left\|B^\top X\right\|_F^2$. This is what the rest of the problem is about.

(d) Now, we are going to zoom in on a special case of our main theorem first. Consider the special case of $X = \Sigma$ matrices (again, X is wide with m rows and $n \geq m$ columns) that are already diagonal. Further suppose that the diagonal of Σ is non-negative and sorted so that it has $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m \geq 0$ down the diagonal. (i.e. we are considering the kinds of Σ matrices that the SVD gives us.)

To warm up, further restrict attention to matrices B that are made up of only standard basis vectors (i.e. each of the k columns of B has exactly one 1 in them and the rest of that column is zero.) Furthermore, to be orthonormal, no two columns can have a 1 in the same row. These constraints mean that B is like the identity matrix $I \in \mathbb{R}^{m \times m}$, except with $m - k$ columns removed and the remaining k columns possibly shuffled.

Under that assumption, **show that for such a B, that $\left\|B^\top \Sigma\right\|_F^2$ must be a sum of k different σ_i^2.**

(HINT: Realize that each of the columns of B basically will pick out exactly one of the σ_i.)

(e) Building on the previous part and its very special assumptions, **show that any resulting $BB^\top \Sigma$ is going to be a diagonal matrix and have the i-th diagonal entry equal to either 0 or σ_i.**

(HINT: The i-th standard basis vector $\vec{e}_i$ is either one of the columns of B, or it is not. What happens if it is one of the columns of B? What happens if it's not one of the columns of B?)

(f) Building on the previous parts and their very special assumptions, **show that a best such B for minimizing $\left\|\Sigma - BB^\top \Sigma\right\|_F^2$ is such that $\sigma_1, \sigma_2, \ldots, \sigma_k, 0, \ldots, 0$ is the diagonal of $BB^\top \Sigma$.**

Further assume that all of the $\sigma_i > \sigma_k$ for $i < k$. In other words, the singular values are strictly decreasing. (This is just to prevent ties that wouldn't change anything, but would slightly complicate writing the proof.)

(HINT: You may cite Homework 3 Problem 8. There's a reason why we assigned that problem to you. What do you want to maximize here?)

(g) Now we have gotten warmed up by dealing with the very special case with the artificial restriction that B has to be made of standard basis vectors. Now, we remove that artificial restriction. From

this point forward, B is a generic matrix with k orthonormal columns. They can be anything we want. Let's look at the columns $\vec{c}_i$ of $C = B^\top$. We can immediately see that $B^\top\Sigma = C\Sigma = \begin{bmatrix}\sigma_1\vec{c}_1 & \sigma_2\vec{c}_2 & \cdots & \sigma_m\vec{c}_m & \vec{0} & \cdots & \vec{0}\end{bmatrix}$.

So we need to get a handle on these columns. Since $\|C\|_F^2 = \|B\|_F^2 = k$, we know that $\sum_{i=1}^n \|\vec{c}_i\|^2 = k$. We also know that since they are norms, that each of the $\|\vec{c}_i\|^2 \geq 0$.

Show that $\|\vec{c}_i\|^2 \leq 1$ **for every** $i = 1, \ldots, m$**.**

(HINT: Invoke Gram-Schmidt to assert that you can extend B with $m-k$ more orthonormal vectors to get a square orthonormal matrix $\widetilde{B}$. Then, since $\widetilde{B}^\top\widetilde{B} = \widetilde{B}\widetilde{B}^\top = I$, what do you know about the norms of the columns of $\widetilde{B}^\top$? What is the relationship of those norms to the norms of $\vec{c}_i$?)

(h) You can see from above that $\left\|B^\top\Sigma\right\|_F^2 = \sum_{i=1}^m \sigma_i^2\|\vec{c}_i\|^2$ and that further $0 \leq \|\vec{c}_i\|^2 \leq 1$ for each i and their sum $\sum_{i=1}^m \|\vec{c}_i\|^2 = k$.

Further assume that all of the $\sigma_i > \sigma_j$ for $i < j$. (This is just to prevent ties that wouldn't change anything, but would complicate writing the proof.)

Show that under those constraints, $\sum_{i=1}^m \sigma_i^2\|\vec{c}_i\|^2 \leq \sum_{i=1}^k \sigma_i^2$**.**

(Hint: Again, you may cite Homework 3 Problem 8. There is a reason we made you do that earlier.).

Because you know this bound can be hit, you have actually proved that the best k-dimensional subspace for approximating Σ in Frobenius norm is just that spanned by the first k standard basis vectors. In other words, the best rank k approximation to a diagonal matrix Σ with non-negative elements $\sigma_i \geq 0$ on the diagonal that are non-increasing (i.e. $\sigma_i \geq \sigma_j$ if $i < j$) is the diagonal matrix with $\sigma_1, \sigma_2, \ldots, \sigma_k$ on the diagonal at the beginning and zero everywhere else.

(i) We have already proved in Homework 11 that using the SVD, the Frobenius norm can be understood in terms of the singular values, i.e., $\|A\|_F^2 = \sum_{i=1}^{\min(m,n)} \sigma_i^2$, where σ_i's are the singular values of A.

Now **solve for** $\widehat{X}$ in equation (1) using the results you developed so far in earlier parts of this problem for the diagonal case.

(HINT: The SVD of X is going to be useful here.)

Congratulations! You have now been walked (hopefully not dragged!) through an elementary proof of why the SVD gives you the best low-rank approximation to a matrix of data. This is the heart of PCA.

In the linear-algebraic (and machine learning) literature, this is called the Eckhart-Young-Mirsky Theorem but all of the proofs that are easily accessible online or in standard textbooks take a more challenging (but shorter) route to the result. The argument given here is in more elementary 16AB style — it is the figurative long "green circle" trail down from the top of the mountain as compared to the black diamond path taken by more experienced skiers. Anyway, this important result justifies many uses of the SVD in machine learning, control, and statistics.

3. Inverse Kinematics

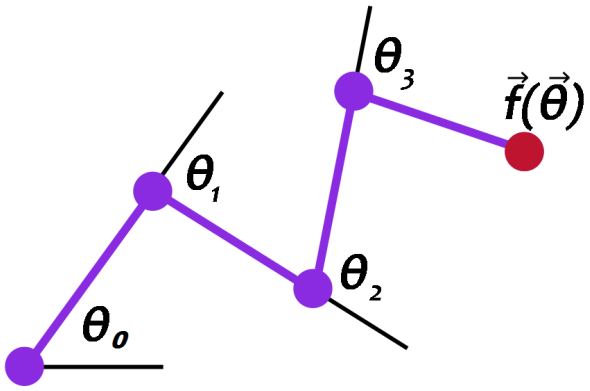

Figure 1: An example of an arm parameterized by θ_0, θ_1, θ_2, and θ_3 with the end effector at point $\vec{f}(\vec{\theta})$.

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:

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}. \tag{5}$$

Suppose further that we have some target point $\vec{t} \in \mathbb{R}^2$, 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. From physics and kinematics, we can find the function $\vec{f}(\vec{\theta})$ that given the angles of each joint can return the position of the end effector. Figure 1 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

$$\vec{g}(\vec{\theta}) = \vec{f}(\vec{\theta}) - \vec{t} \tag{6}$$

is equal to $\vec{0}$.

Note that this would be simple to do if $\vec{f}$ had an inverse. However due to physics and rotations, many sines and cosines appear in the forward kinematics and $\vec{f}$ becomes highly nonlinear. Inverse kinematics is the problem of given this point in space that we want to reach, what should we set the joint angles of our arm to? This ends up being a crucial problem to solve, and inverse kinematics appears everywhere in robotics, control, and computer graphics applications. To learn more about how to solve these problems, we recommend taking EECS 106A, but we will now provide one such imperfect way to solve this problem.

To accomplish this, we use the spirit of Newton's method for solving potentially nonlinear equations. You might have seen Newton's method for finding roots of scalar functions in your calculus course. In this 1-D case, you have a real scalar function g of a single parameter θ and we want to find a $\widehat{\theta}$ so that $g(\widehat{\theta}) = 0$.

Step i of Newton's method does the following, where our current estimate of $\widehat{\theta}$ is $\theta^{(i)}$:

1. Linearize g around $\theta^{(i)}$ to get an approximation $\widetilde{g}$:

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

2. We want to find the roots of g, and will get closer by finding the roots of this linear approximation $\widetilde{g}$. Thus we want to set $\widetilde{g}(\theta) = 0$ to get

$$0 = g(\theta^{(i)}) + g'(\theta^{(i)})(\theta - \theta^{(i)}) \tag{8}$$

$$\theta = \theta^{(i)} - \frac{g(\theta^{(i)})}{g'(\theta^{(i)})} \tag{9}$$

3. Therefore for the next iteration, we set $\theta^{(i+1)} = \theta^{(i)} - \frac{g(\theta^{(i)})}{g'(\theta^{(i)})}$, and repeat.

We will iterate this until $g(\theta)$ is close enough to 0 for our application. In practice, instead of solving exactly for $g(\theta) = 0$, 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 $g'(\theta^{(i)})$ might be very different from $g'(\theta^{(i+1)})$. After all, linearization is only valid in a local neighborhood. (Isn't it interesting that you've already seen the idea of just taking small steps before? You saw it in the problem where we iteratively solved least-squares problems. So 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 $\vec{g}(\vec{\theta})$:

$$J_{\vec{g}} = \begin{bmatrix} \frac{\partial g_x(\vec{\theta})}{\partial \theta_1} & \frac{\partial g_x(\vec{\theta})}{\partial \theta_2} & \frac{\partial g_x(\vec{\theta})}{\partial \theta_3} & \frac{\partial g_x(\vec{\theta})}{\partial \theta_4} \\ \frac{\partial g_y(\vec{\theta})}{\partial \theta_1} & \frac{\partial g_y(\vec{\theta})}{\partial \theta_2} & \frac{\partial g_y(\vec{\theta})}{\partial \theta_3} & \frac{\partial g_y(\vec{\theta})}{\partial \theta_4} \end{bmatrix}. \tag{10}$$

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.

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

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta J_{\vec{g}}^{\dagger}(\vec{\theta}^{(i)})\vec{g}(\vec{\theta}^{(i)}) \tag{11}$$

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 now know how to "invert" any matrix, using the Moore-Penrose pseudoinverse that you saw in a previous homework. The minimality property of the Moore-Penrose pseudoinverse that we saw is extremley 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 pseudoinverse algorithm are:

1. First, compute the compact-form SVD of the input matrix.
2. Next, we compute Σ^{-1} by inverting each nonzero singular value σ_i. We assume that any singular value less than some ϵ is the same as 0, since inverting small values will cause numerical issues.
3. 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 use them correctly. For example, for this problem you want the compact SVD so what argument should you call `svd` with? 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}.** To be careful of numerical issues, first threshold the singular values, and only invert the singular values above a certain value ϵ, considering smaller ones as 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. Considering small singular values as being 0 stops this from happening.

(c) We now have all of the parts to compute the relevant pseudoinverse of A. **Finish the last line of the function to calculate the pseudoinverse.**

(HINT: `np.diag` *can be a very useful tool in converting a vector into a square diagonal matrix. Also remember to use* `.T` *to get 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.

Finally, there is a test case where you can drag the target position and the robot joints will update automatically as you track the target. You can also click anywhere on the plot and the robot will attempt to reach it.

Describe what you see happening in the animation as well as the plot where you drag the target position.

4. 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.

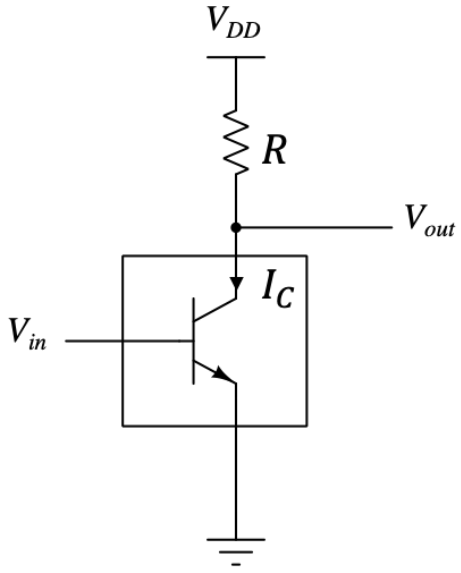

Figure 2: Voltage amplifier circuit using a BJT

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:

$$I_C(V_{\text{in}}) = I_S \cdot e^{\frac{V_{\text{in}}}{V_{\text{TH}}}}, \tag{12}$$

where V_{TH} is the thermal voltage. We can assume $V_{\text{TH}} = 26\,\text{mV}$ 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 $I_C(V_{\text{in}})$ holds as long as $0.2\,\text{V} < V_{\text{out}} < 40\,\text{V}$, 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 $(V_{\text{in}}^\star, V_{\text{out}}^\star)$ 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 $V_{\text{in}} = V_{\text{in}}^\star + \delta V_{\text{in}}$ and $V_{\text{out}} = V_{\text{out}}^\star + \delta V_{\text{out}}$, then we want the magnitude of the 'amplification gain' given by $\left|\frac{\delta V_{\text{out}}}{\delta V_{\text{in}}}\right|$ to be large. We're going to investigate this amplification using linearization.

(NOTE: in this problem, δV is single variable indicating a small variation in V, not $\delta \times V$.)

(a) **Write a symbolic expression for V_{out} as a function of I_C, V_{DD} and R in Fig 2.**

(b) Now let's linearize I_C in the neighborhood of an input voltage $V_{\text{in}}^\star$ and a specific $I_C^\star$. Assume that you have a found a particular pair of input voltage $V_{\text{in}}^\star$ and current $I_C^\star$ that satisfy the current equation (12).

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:

$$I_C(V_{\text{in}}) = I_C(V_{\text{in}}^\star) + m(V_{\text{in}} - V_{\text{in}}^\star) = I_C^\star + m\,\delta V_{\text{in}} \tag{13}$$

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

What is m here as a function of $I_C^\star$ 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, $\delta I_C = m\,\delta V_{\text{in}}$ around a known solution $(I_C^\star, V_{\text{in}}^\star)$. 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 $(V_{\text{in}}^\star, V_{\text{out}}^\star)$ 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 $V_{\text{in}} = V_{\text{in}}^\star + \delta V_{\text{in}}$ and $V_{\text{out}} = V_{\text{out}}^\star + \delta V_{\text{out}}$, then we want the magnitude of the 'amplification gain' given by $\left|\frac{\delta V_{\text{out}}}{\delta V_{\text{in}}}\right|$ to be large.**

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

$$V_{\text{out}}^\star = V_{\text{DD}} - RI_C^\star$$

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

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

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

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

(e) If $I_C^\star = 9\,\text{mA}$ when $V_{\text{in}}^\star = 0.7\,\text{V}$ with all other parameters remaining fixed, **verify that the magnitude of the small-signal voltage gain $\left|\frac{\delta V_{\text{out}}}{\delta V_{\text{in}}}\right|$ 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 $V_{\text{in}}^\star = 0.65\,\text{V}$ (part d) and $V_{\text{in}}^\star = 0.7\,\text{V}$ (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.

5. Single-dimensional linearization

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

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = \sin\big(x(t)\big) + u(t). \tag{14}$$

(a) The first thing we want to do is find equilibria of the system. Recall that these are the values of (x, u) such that the derivative of x is 0. Suppose we want to investigate potential expansion points $(x^\star, u^\star)$ with $u^\star = 0$. **Sketch** $\sin(x^\star)$ **for** $-4\pi \leq x^\star \leq 4\pi$ **and intersect it with the horizontal line at** 0**.** This will show us the equilibria points, where $\sin(x^\star) + u^\star = 0$.

(b) **Show that all** $x(t) = x^\star_m = m\pi$ **satisfy** (14) **together with** $u^\star = 0$**, i.e.** $u(t) = u^\star = 0\ \forall t$**.**

Let us zoom in on two choices: $x^\star_{-1} = -\pi$ and $x^\star_0 = 0$. Looking at the sketch we made, these seem like representative points.

(c) Linearize the system (14) around the equilibrium $(x^\star_0, u^\star) = (0, 0)$. **What is the resulting linearized scalar differential equation for** $x_\ell(t) = x(t) - x^\star_0 = x(t) - 0$**, involving** $u_\ell(t) = u(t) - u^\star = u(t) - 0$**?** 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 $(x^\star, u^\star)$. 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 (14) around the equilibrium $(x^\star_{-1}, u^\star) = (-\pi, 0)$. **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 $(x^\star, u^\star)$. 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

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

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

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

6. 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.

7. 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:

- Yuxun Zhou.
- Anant Sahai.
- Druv Pai.
- Rahul Arya.

- Moses Won.
- Aditya Arun.
- Pavan Bhargava.
- Stephen Bailey.
- Ashwin Vangipuram.
- Kris Pister.
- Alex Devonport.
- Regina Eckert.

