EECS 16B Designing Information Devices and Systems II
Fall 2019 UC Berkeley HW 12

1. Low Rank Approximation of a Matrix

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

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

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

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.* (The analogous story could be told about rows, but we'll keep our focus on columns since you are all more comfortable working with columns from 16A and 16B so far.)

To understand this problem, we will have to also think about what it might mean to approximate a matrix and we use the natural matrix norm to do this. 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.

(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 $m \times n$ independent elements (entries). In other words, an arbitrary $m \times n$ matrix has $m \times n$ degrees of freedom. **How much information (independent elements/degree of freedom) do we have to know to specify a rank r matrix of the same $m \times n$ size. How is low-rank approximation a kind of lossy compression for a matrix?**
(HINT: Think about outer-product representations for a rank r matrix, for example, that given by the SVD.)

(b) Now, let us start into understanding the approximation itself. Before we get into the "hidden subspace" aspect, we need to think about what it means to approximate. Suppose we view a matrix A as a list of columns:

$$A = [\vec{a}_1, \vec{a}_2, \cdots, \vec{a}_n] \tag{3}$$

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

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

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. (It turns out the same thing is true for viewing it as a collection of rows.)

(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. 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 for it. (Note here that $B \in \mathbb{R}^{m \times k}$.)

Before we worry about finding such a basis or subspace, it is good to understand how we would approximate X using it. But we know how to approximate a column in a subspace! We can project into it. So we can project all of X into that subspace by computing BB^TX. To understand the quality of this approximation, first **show that** $\|X - BB^TX\|_F^2 = \|X\|_F^2 - \|BB^TX\|_F^2$.

(HINT: Give things names and invoke the Pythagorean theorem. Let $\vec{x}_i$ be the i-th column of X and let $Y_B = BB^TX$, and let $\vec{y}_{B,i}$ be the i-th column of Y_B. Let $R_B = X - Y_B$ be the residual that remains when estimating X using the basis B. Then show the desired result for each column using the properties of projection and put things together.)

Now, our goal is to find the optimal such basis B. We can see from the previous part that since $\|X\|_F^2$ is outside our control, minimizing $\|X - BB^TX\|_F^2$ is the same as finding a matrix B with k orthonormal columns that maximizes $\|BB^TX\|_F^2$. Homework 10 showed that for any matrix A, and a matrix with orthonormal columns U:

$$\|UA\|_F = \|A\|_F.$$

Using this fact, it immediately follows that: $\|BB^TX\|_F^2 = \|B^TX\|_F^2$. Consequently, it suffices to find a B with orthonormal columns that maximizes $\|B^TX\|_F^2$. 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 (with n rows and $m \geq n$ columns) that are already diagonal. Further suppose that the diagonal of Σ is non-negative and sorted so that it has $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ 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. these are the columns of the identity — each of the k columns of B has at most 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 — we have to pick a subset of the columns of the identity matrix as our B matrix.

Under that assumption, **show that for such a B, the $\|B^TX\|_F^2$ 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 $Y_B = BB^T\Sigma$ 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 is either in the basis B or not. What happens if it is in the basis B? What happens if it is not in the basis B?)

(f) Building on the previous part and its assumptions, **show that a best such $Y_B = BB^T\Sigma$ (for minimizing $\|\Sigma - Y_B\|_F^2$) has $\sigma_1, \sigma_2, \ldots, \sigma_k, 0, \ldots, 0$ down the diagonal.**

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

(HINT: In a sense, this should feel a bit obvious. But how do you prove it? The previous parts tell you that we want to maximize $\|B^T\Sigma\|_F^2$, which in turn is the sum of the squares of any k distinct diagonal elements of Σ. Now, the claim is that choosing the largest k of the σ's will give the largest sum. As is often the case for obvious things, it is easiest to proceed by a contradiction argument. Suppose that you had selected a truly different subset of the σs. How do you show that this subset isn't the best possible subset? One way is to show how you can strictly improve on it. What would you do to improve on it?)

(g) Now that we have dealt with the very special case as a warmup, we want to relax our artificial restriction that B has to be made of standard basis vectors. Let's look at the columns $\vec{c}_i$ of $C = B^T$. We can immediately see that $B^T\Sigma = C\Sigma = [\sigma_1\vec{c}_1, \sigma_2\vec{c}_2, \cdots, \sigma_n\vec{c}_n, \vec{0}, \vec{0}, \cdots, \vec{0}]$.

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 $i = 1, \ldots, n$.

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

(h) You know from above that $\|B^T\Sigma\|_F = \sum_{i=1}^n \sigma_i^2\|\vec{c}_i\|^2$ and that further $0 \leq \|\vec{c}_i\|^2 \leq 1$ for each i and their sum $\sum_{i=1}^n \|\vec{c}_i\|^2 = k$.

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

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

(Hint: For convenience in writing, give new names $f_i = \|\vec{c}_i\|^2$ and remember that $\sum f_i = k$ while $0 \leq f_i \leq 1$.

This is another thing that should feel like it is kind of obvious. It should feel like a more continuous version of what you already showed earlier (two parts ago) about the best possible subset of numbers to sum up if we want to maximize the sum — pick the biggest numbers. Now, you're allowed to take fractional parts of the numbers if you'd like. You probably have no doubt that the best thing to do is to just take the biggest numbers. Why would you want to take half of a big number and half of a smaller number instead of all of the big number?

Anyway, you need to make this thinking into a proof. Getting an upper bound is about maximizing. You know how to reach that bound. Assume that you have some different allocation (i.e. it includes some positive allocation in an index $> k$) of the k total weight across the f_i that is claimed to be optimal and bigger than your claimed bound. Show that you can make it even bigger by redistributing the weight while keeping your constraints all satisfied. This contradicts the supposed optimality of the claimed optimal allocation. And so no such different allocation can exist.)

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-decreasing (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 a previous homework that using the SVD, the Frobenius norm can be understood in terms of the singular values $\|A\|_F^2 = \sum_{i=0}^{\min(m,n)} \sigma_i^2$ (Recall that a $m \times n$ matrix can have at most $\min(m,n)$ nonzero singular values), where σ_i's are the singular values of A.

This was proved as a consequence of the fact that if U has orthonormal columns, then $\|UA\|_F^2 = \mathrm{Tr}(A^TU^TUA) = \mathrm{Tr}(A^TIA) = \mathrm{Tr}(A^TA) = \|A\|_F^2$ even if U is not square, and similarly if V^T has orthonormal rows, then $\|AV^T\|_F^2 = \mathrm{Tr}(AV^TVA^T) = \mathrm{Tr}(AIA^T) = \mathrm{Tr}(AA^T) = \|A\|_F^2$. That means that the SVD $A = U\Sigma V^T$ implies that $\|A\|_F = \|\Sigma\|_F$.

Now **please solve for $\hat{X}$** 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.

2. Controllable Canonical Form and Eigenvalue Placement

Consider a linear discrete time system below ($\vec{x} \in \mathbb{R}^n$, $u \in \mathbb{R}$, and $\vec{b} \in \mathbb{R}^n$).

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

If the system is *controllable*, then there exists a transformation $\vec{z} = T^{-1}\vec{x}$ (where T is the invertible matrix whose columns are the basis vectors for the new space) such that in the transformed coordinates, the system is in *controllable canonical form*, which is given by

$$\vec{z}(t+1) = \widetilde{A}\vec{z}(t) + \widetilde{\vec{b}}u(t) = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ a_0 & a_1 & a_2 & \dots & a_{n-1} \end{bmatrix} \vec{z}(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} u(t)$$

Here, $\widetilde{A} = T^{-1}AT$ and $\widetilde{\vec{b}} = T^{-1}\vec{b}$.

The characteristic polynomials of the matrices A and $\widetilde{A}$ are the same and given by

$$\det(\lambda I - A) = \det(\lambda I - \widetilde{A}) = \lambda^n - a_{n-1}\lambda^{n-1} - a_{n-2}\lambda^{n-2} - \cdots - a_0. \tag{5}$$

(a) **Directly show that A and $\widetilde{A}$ have the same eigenvalues.**
(*Hint: let $\vec{v}$ be an eigenvector of A; use $T^{-1}\vec{v}$ for $\widetilde{A}$*)

(b) Let the controllability matrices (what we use to check controllability) C and $\widetilde{C}$ be $C = \begin{bmatrix} \vec{b} & A\vec{b} & \cdots & A^{n-1}\vec{b} \end{bmatrix}$ and $\widetilde{C} = \begin{bmatrix} \widetilde{\vec{b}} & \widetilde{A}\widetilde{\vec{b}} & \dots & \widetilde{A}^{n-1}\widetilde{\vec{b}} \end{bmatrix}$, respectively. **Show that $\widetilde{C} = T^{-1}C$.**

(HINT: Write out $\widetilde{C}$ and substitute in $T^{-1}AT$ for $\widetilde{A}$ and $T^{-1}\vec{b}$ for $\widetilde{\vec{b}}$. Then, cancel $T^{-1}T$ terms where you can. Then, remember what you were taught in 16A about what matrix multiplication means in terms of matrix vector multiplication — i.e. multiplying DF is just a matrix whose i-th column is $D\vec{f}_i$ where $\vec{f}_i$ is the i-th column of F.)

(c) **Use the previous part and the fact that the controllability matrix C is full rank and hence invertible to show that $T^{-1} = \widetilde{C}C^{-1}$.**

(d) **Play with the included jupyter notebook and comment on the difficulty of setting controller gains by hand relative to using the CCF.**

Now, consider the specific system

$$\vec{x}(t+1) = A\vec{x}(t) + \vec{b}u(t) = \begin{bmatrix} -2 & 0 \\ -3 & -1 \end{bmatrix} \vec{x}(t) + \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} u(t) \tag{6}$$

(e) **Show that the system (6) is controllable.**

Since the system is controllable, there exists a transformation $\vec{z} = T^{-1}\vec{x}$ such that

$$\vec{z}(t+1) = \widetilde{A}\vec{z}(t) + \widetilde{\vec{b}}u(t) = \begin{bmatrix} 0 & 1 \\ a_0 & a_1 \end{bmatrix} \vec{z}(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \tag{7}$$

and the characteristic polynomials of the matrices A and $\widetilde{A}$ are the same.

(f) **Compute the matrix $\widetilde{A}$.**

(g) **Compute the controllability matrices $C = \begin{bmatrix} \vec{b} & A\vec{b} \end{bmatrix}$ and $\widetilde{C} = \begin{bmatrix} \widetilde{\vec{b}} & \widetilde{A}\widetilde{\vec{b}} \end{bmatrix}$.**

(h) **Compute the transformation matrix $T^{-1} = \widetilde{C}C^{-1}$.**

(i) **Show that the system (6) is *unstable* in open-loop.** (That is, when we do not apply closed-loop feedback.)

Now, we want to make the system *stable* by applying state feedback for the system in its original (6) and canonical (7) forms.

That is, let $u(t)$ be $u(t) = -\widetilde{\vec{k}}^T \vec{z}(t) = \begin{bmatrix} -\widetilde{k}_0 & -\widetilde{k}_1 \end{bmatrix} \vec{z}(t)$. After applying state feedback, the systems (6) and (7) have the form

$$\vec{x}(t+1) = A_{cl}\vec{x}(t)$$
$$\vec{z}(t+1) = \widetilde{A}_{cl}\vec{z}(t)$$

(j) If we want to apply the same feedback control law directly using the original $\vec{x}(t)$ state, call the resulting law $u(t) = -\vec{k}^T\vec{x}(t) = \begin{bmatrix} -k_0 & -k_1 \end{bmatrix} \vec{x}(t)$. **Give an expression for $\vec{k}$ in terms of $\widetilde{\vec{k}}$ and T.**

(k) **Compute A_{cl} and $\widetilde{A}_{cl}$ in terms of $\widetilde{k}_0$ and $\widetilde{k}_1$.**

(l) **Compute $\widetilde{\vec{k}}$ so that $\widetilde{A}_{cl}$ has eigenvalues $\lambda = \pm\frac{1}{2}$** (*Hint: use the characteristic polynomial (5).*)

(m) Using the corresponding $\vec{k}$ you derived in the previous part, **show that A_{cl} also has eigenvalues $\lambda = \pm\frac{1}{2}$ by explicit calculation.** (Feel free to use numpy to do this.)

3. Single-dimensional linearization

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

$$\frac{d}{dt}x(t) = \sin\big(x(t)\big) + u(t). \tag{8}$$

(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** (8) **together with** $u^* = 0$**.**

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 (8) 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** $u_\ell(t) = u(t) - u^* = u(t) - 0$**?** Don't forget to include a disturbance term that captures the approximation error due to linearization.

(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 over duration Δ, that 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.** This model is an approximation of what will happen if we actually applied a piecewise constant control input to the original nonlinear differential equation.

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

(f) Now linearize the system (8) 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 a disturbance term to capture the approximation error.

(g) For the linearized 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 over duration Δ, that 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.** This model is an approximation of what will happen if we actually applied a piecewise constant control input to the original nonlinear differential equation.

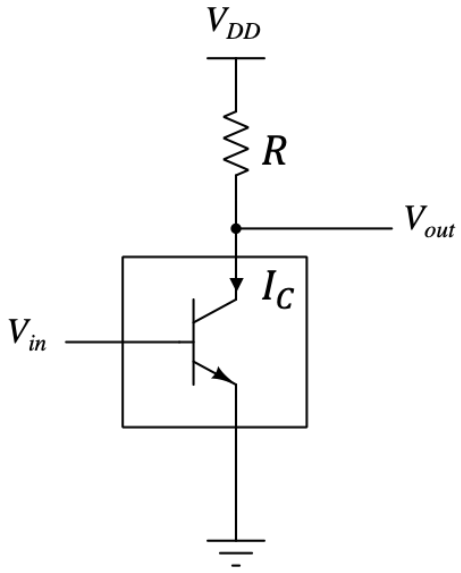
(h) **Is the (approximate) discrete-time system you found in the previous part stable or unstable?**

(i) Suppose for the two *discrete-time systems* above, we chose to apply a feedback law $u(t) = -k(x(t) - x^*)$. **for what range of** k **values, would the resulting linearized discrete-time systems be stable?** Your answer will depend on Δ.

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).



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

$$I_C(V_{in}) = I_S e^{\frac{V_{in}}{V_{TH}}} \tag{9}$$

where V_{TH} is the thermal voltage. We can assume $V_{TH} = 26$ mV at temperatures of 300K (close to room temperature).

With this amplifier, small variations in the input voltage V_{in} can turn into large variations in the output voltage V_{out} under the right conditions. We're going to investigate this amplification using linearization.

Let's consider the 2N3904 transistor, 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.)

(a) **Write a symbolic expression for V_{out} as a function of I_C.**

(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 (9). 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:

$$I_C(V_{in}) = I_C(V_{in}^*) + \delta I_C \approx I_C^* + m(V_{in} - V_{in}^*) = I_C^* + m\ \delta V_{in} \tag{10}$$

where $\delta V_{in} = V_{in} - V_{in}^*$ is the change in input voltage and $\delta I_C = I_C - I_C^*$ is the change in collector current. **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 special properties 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 function is to get the nonlinearity to behave the way we want it to.)

Going back to your equation from part (a), plug in your linearized equation for I_C. Define the appropriate V_{out}^* 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 $V_{DD} = 10V$, $R = 1k\Omega$, and $I_C^* = 1mA$ when $V_{in}^* = 0.65V$, **what is the small-signal voltage gain $\frac{\delta V_{out}}{\delta V_{in}}$, between the input and the output around this bias point?** (one or two digits of precision is plenty)

(e) If $I_C^* = 9mA$ when $V_{in}^* = 0.7V$, **what is the small-signal voltage gain around this bias point?** (one or two digits is plenty)

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. 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.

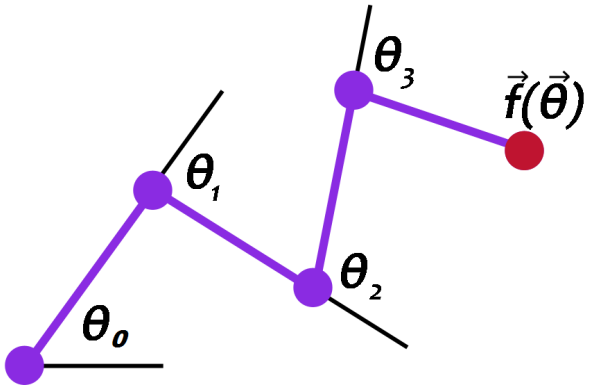


Figure 1: An example of an arm parameterized by θ_0, θ_1, θ_2, and θ_3 with the end effector at point $\vec{f}(\vec{\theta})$.

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:

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}. \tag{11}$$

Suppose further that we have some target $\vec{t} \in \mathbb{R}^2$, wich 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 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

$$\vec{\theta} \in \mathbb{R}^4 \mapsto \vec{g}(\vec{\theta}) = \vec{f}(\vec{\theta}) - \vec{t} \tag{12}$$

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

(b) θ^{i+1} solves $\hat{g}^i(\theta) = 0$. Note that this is easy to solve, since $\hat{g}^i$ is linear and 1-d. Note also that doing this is solving: $g'(\theta^{(i)})(\theta - \theta^{(i)}) = g(\theta^{(i)})$, which reduces to "inverting" the linear operator $z \mapsto g'(\theta^{(i)})z$.

and 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 θ^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 θ^{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 $\vec{g}(\vec{\theta})$:

$$J_{\vec{g}} = \begin{bmatrix} \frac{\partial g_x(\vec{\theta})}{\partial \theta_1} & \frac{\partial g_x(\vec{\theta})}{\partial \theta_2} & \frac{\partial g_x(\vec{\theta})}{\partial \theta_3} & \frac{\partial g_x(\vec{\theta})}{\partial \theta_4} \\ \frac{\partial g_y(\vec{\theta})}{\partial \theta_1} & \frac{\partial g_y(\vec{\theta})}{\partial \theta_2} & \frac{\partial g_y(\vec{\theta})}{\partial \theta_3} & \frac{\partial g_y(\vec{\theta})}{\partial \theta_4} \end{bmatrix}. \tag{13}$$

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

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta J_{\vec{g}}^{-1}(\vec{\theta}^{(i)})\vec{g}(\vec{\theta}^{(i)}) \tag{14}$$

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.

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.

(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, which form of the SVD do you want? 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.)*

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 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.

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

(c) **How did you work on this homework?** (For example, *I first worked by myself for 2 hours, but got stuck on problem 3, so I went to office hours. Then I went to homework party for a few hours, where I finished the homework.*)

(d) **Roughly how many total hours did you work on this homework?**

Contributors:

- Yuxun Zhou.
- Anant Sahai.
- Rahul Arya.
- Moses Won.
- Aditya Arun.
- Pavan Bhargava.
- Yen-Sheng Ho.
- Nathan Lambert.
- Kris Pister.
- Alex Devonport.
- Regina Eckert.
- Stephen Bailey.

