CSE 512: Challenge!

Homework 1

Generalizing the Cauchy Schwartz inequality

We saw in lecture the Cauchy Schwartz inequality, which says that for any two vectors $x \in \mathbb{R}^n$, $y \in \mathbb{R}^n$,

$$x^T y \le \|x\|_2 \|y\|_2, \quad x^T y = \|x\|_2 \|y\|_2 \iff x = cy$$

for some positive scalar c.

1. Holder's inequality generalizes this to *dual norms.* That is, for any p and q where $\frac{1}{p} + \frac{1}{q} = 1$, we have

$$x^T y \le \|x\|_p \|y\|_q$$

where equality is reachable for a specific choice of x and y.

(a) When $p = 1$, the corresponding q is $q = +\infty$, and $\|x\|_\infty$ is the max-norm, e.g. $\|x\|_\infty = \max_i |x_i|$. Prove Holder's inequality for this choice of p and q. That is, prove that for any x and any y,

$$x^T y \le \|x\|_1 \|y\|_\infty.$$

Ans.

$$\begin{aligned} x^T y &= \sum_{i=1}^n x[i]y[i] \\ &\le \sum_{i=1}^n |x[i]| \cdot |y[i]| \\ &\le \sum_{i=1}^n |x[i]| \cdot \max_j |y[j]| \\ &= \|x\|_1 \|y\|_\infty \end{aligned}$$

(b) For $p = 1$, $q = +\infty$, given x, list the entire set of possible y where $x^T y = \|x\|_1 \|y\|_\infty$. Hint: The rule for $x = [0, 0, ..., 0]^T$ is different than for x if all the values are nonzero.

Ans. Note the two areas where we have inequalities. First, the signs of the two indices must match whenever $x[i] \neq 0$:

$$\mathbf{sign}(y[i]) = \mathbf{sign}(x[i]) \quad \forall i : x[i] \neq 0.$$

Second, it must be that whenever $x[i] \neq 0$ then $\mathbf{sign}(x[i])y[i] = \max_j |y[j]|$:

$$y[i] \cdot \mathbf{sign}(x[i]) = \max_j |y[j]| \quad \forall i : x[i] \neq 0.$$

This second rule in fact summarizes all the possible y such that Holder's inequality is tight.

2. We can further generalize Holder's inequality to include *conic constraints.* For example, suppose x is restricted to the nonnegative orthant, e.g. $x_i \ge 0$ for all i. Then

$$x^T y \le \left(\sum_i x_i\right) \max_i y_i. \qquad (*)$$

(a) Prove $(*)$. **Ans.** Actually, the proof is just a byproduct of the previous proof, and can be written succinctly as

$$\begin{aligned} x^Ty &= \sum_{i=1}^n x[i]y[i] \\ &\le \sum_{i=1}^n (x[i]\cdot \max_j y[j]) \\ &= \left(\sum_{i=1}^n x[i]\right)\cdot\left(\max_j y[j]\right) \end{aligned}$$

(b) List the set of y such that, given x, $(*)$ is true with equality.
Ans. Just following the same logic as in the previous problem, the set of y must satisfy

$$y[i] = \max_j y[j] \quad \forall i : x[i] > 0.$$

3. The *singular value decomposition* of a matrix $X \in \mathbb{R}^{m\times n}$ decomposes X to its *singular values and vectors*. It is usually written as

$$X = \sum_{i=1}^r s_i u_i v_i^T$$

where $u_i \in \mathbb{R}^m$ are the *left singular values*, $v_i \in \mathbb{R}^n$ are the *right singular values*, and s_i are positive scalars. Here, r is the *rank* of X. Each singular vector are *orthonormal*, e.g.

$$u_i^Tu_j = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{else,}\end{cases} \qquad v_i^Tv_j = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{else.}\end{cases}$$

The *trace norm* of X is the sum of its singular values (denoted $\|X\|_*$). The *spectral norm* of X is its largest singular value (denoted $\|X\|_2$).

(a) Prove the following generalization of the Cauchy Schwartz inequality for matrices $X \in \mathbb{R}^{m\times n}$ and $Y \in \mathbb{R}^{m\times n}$

$$\mathbf{tr}(X^TY) \le \|X\|_*\|Y\|_2.$$

Ans. Take the singular value decomposition of

$$X = \sum_{i=1}^r \sigma_i u_i v_i^T, \qquad Y = \sum_{i=1}^r \sigma_i' u_i' v_i'^T.$$

There are two main things to prove here:

i. $\mathbf{tr}(X^TY) \le \sum_i \sigma_i\sigma_i'$, e.g. the trace is maximized if $u_i = u_i'$ and $v_i = v_i'$ for all i, and
ii. $\sum_i \sigma_i\sigma_i' \le \|X\|_*\|Y\|_2$, which is a rehash of parts 1 and 2.

The first one was much more nontrivial than I originally thought. I did not take points away if all you got was the second part, if there was any kind of reasonable attempt to the first part.

There are probably more than one way of doing the first part, but here's the way that to me required the least amount of prior linear algebra (but still requires some).

It actually makes things easier to express the SVDs in matrix form, e.g.

$$X = U\Sigma V^T, \qquad Y = U'\Sigma' V'^T.$$

We now think of the two parts as first, optimizing over the singular vectors (U, V, U', V'), and then optimizing over the singular values (Σ, Σ'). We write

$$\mathbf{tr}(X^TY) = \mathbf{tr}(V\Sigma^TU^TU'\Sigma'V'^T) \overset{\mathbf{tr}(AB)=\mathbf{tr}(BA)}{=} \mathbf{tr}(\Sigma^TU^TU'\Sigma'V'^TV).$$

I think the easiest way to continue from here is to use some prior linear algebra knowledge. Specifically, we may know that for any orthonormal matrix Q, for any matrix A, that $\mathbf{tr}(AQ) = \mathbf{tr}(QA) = \mathbf{tr}(A)$. So, this simplifies

$$\mathbf{tr}(X^TY) = \mathbf{tr}(\Sigma^T U^T U'\Sigma') \overset{\mathbf{tr}(AB)=\mathbf{tr}(BA)}{=} \mathbf{tr}(U^TU'\Sigma'\Sigma^T) = \mathbf{tr}(\Sigma'\Sigma^T) = \mathbf{tr}(\Sigma^T\Sigma').$$

We can then invoke previous results:

$$\mathbf{tr}(X^TY) \le \sum_{i=1}^r \sigma_i\sigma_i' \le \sum_{i=1}^r |\sigma_i| \cdot \max_{j=1}^r |\sigma_j'| = \|\sigma\|_1\|\sigma'\|_\infty = \|X\|_*\|Y\|_2.$$

From that, we can extrapolate the result.

The "budget problem" way. Let us define $Z = U^TU'\Sigma'V'^TV$. My claim is that

$$\mathbf{tr}(X^TY) = \mathbf{tr}(\Sigma^TZ) \le \sum_i \sigma_i \max_j \sigma_j'.$$

In particular, writing out $z = \mathbf{diag}(Z)$, I can rewrite $\mathbf{tr}(\Sigma^TZ) = \sigma^Tz$, and I have an LP on my hands:

$$\begin{array}{ll} \max_z & \sum_i \sigma_i z_i \\ \text{subject to} & \sum_i z_i = \sum_i \sigma_i' \end{array}$$

Well, if I view this as a budgetting problem, where the nonnegative values σ_i are like prices of, say, candy at a candy shop, and my goal is to max out my mom's income, then I'll pick z_i to put all its weight on the most expensive item such that the budget is preserved. This is achieved with $z_1 = \sum_i \sigma_i'$ and $z_2 = ... = z_n = 0$. This is satisfied with

$$Z = \mathbf{diag}(s, 0, ..., 0)$$

and therefore $\|Z\|_2 = s$. I'm not saying this way is better than the previous solution, but I like it because it requires fewer "tricks" and the budgetting framework gives a bit more intuition.

(b) Given X, describe the set of Y such that $\mathbf{tr}(X^TY) = \|X\|_*\|Y\|_2$.

Ans.

- First, note that, whenever $\sigma_i^X > 0$ or $\sigma_i^Y > 0$, it must be that $u_i^X = u_i^Y$ and $v_i^X = v_i^Y$, to ensure that $\mathbf{tr}(X^TY) = \sum_{i,j}\sigma_i^X\sigma_j^Y$.
- Second, we require the nontrivial singular values of Y (corresponding to these singular vectors) to have max value.
- Overall, the set of Y can be described as

$$\left\{ Y = \sum_{i=1}^r \sigma^Y u_i v_i^T + \sum_{i=r+1}^n \sigma_i^Y u_i v_i^T : X = \sum_{i=1}^r \sigma_i^X u_i v_i^T, \quad \sigma^Y \ge \max_{i=r+1,\dots,n} \sigma_i^Y \right\}$$

where $\sum_{i=1}^r \sigma^Y u_i v_i^T + \sum_{i=r+1}^n \sigma_i^Y u_i v_i^T$ is the singular value decomposition of Y.

Homework 2

(2 pts) Today we have two separate questions, each worth one point.

1. The n-gram version of the classifier from the Alice in Wonderland exercise would be the not-Naive-Bayes version; that is, rather than just counting how many times "mad" precedes "hatter" and "the" is 2 words previous of "hatter", we would like to count exact phrases, e.g. how many times "the mad" precedes "hatter". If we count 1 word ago, we are building a bigram classifier; 2 words is trigram, etc. In general, an n-gram classifier looks for the probabilities of all n-grams, e.g. given a history of $n-1$ words, predict the next best word.

   In this exercise, build an n-gram classifier for "Alice in Wonderland". Note that using a full memory of every possible combination of n words will quickly become memory inefficient, so you want to "leverage sparsity"; that is, only store the phrases you see. Then, you will find that you can extend n quite a bit with no problem.

   What to do:

- Using the n-gram classifier trained on "Alice in Wonderland", **give the train accuracy** (on same corpus) and **test accuracy** (on Alice through the looking glass) for next-word-prediction.
- Sweep the hyperparameters for $n = 2, 3, ..., 100$, and give the train and test accuracy for each value. Do you notice a "best" value of n? At what values of n do you detect overfitting, e.g. train accuracy $\gg$ test accuracy?
- For a good choice of hyperparameter, show the result of 100 words generated using any seed found in either corpus. Do this by sampling the likelihood randomly. Is this new generator any good, in your opinion?

Ans. Solution: see python notebook, attached. Overall, the performance is pretty terrible using large n. I think this could be resolved by doing better data cleaning in the future.

2. **Linearly hashed KNN** If your computer is anything like my dinky laptop, that KNN exercise was a pain to run. As we discussed in class, in general, KNN is tough on runtime memory and computation, because in general we don't like doing things like training during prediction. One way to get around this is **dimensionality reduction**. While there are many clever ways of implementing such a thing, for this exercise we will just try something very simple, yet sometimes shockingly effective: **linear hashing**.

Linear hashing basically follows the principle (sometimes formally stated as the Johnson-Lindenstrauss lemma):

$$\mathbf{dist}(x, y) \approx \mathbf{dist}(Ax, Ay)$$

where $A \in \mathbb{R}^{n\times d}$ is some random matrix with $d \ll n$. That is to say, if I take some points in space, and hit them with a random projection to a lower dimensional space, their configuration should be roughly preserved. We won't really go into this until later lectures, but for now we can evaluate this principle empirically, by creating linearly hashed KNN.

Add a new box in your KNN ipython notebook. Code in two hashing functions:

- **Random subsampling** Pick a random subset $\mathcal{S} \subset \{1, ..., n\}$, $|\mathcal{S}| = d$ and reduce each feature vector so that only the features in $\mathcal{S}$ remain.
- **Random matrix multiply** Generate a random matrix $A \in \mathbb{R}^{d\times n}$ where each A_{ij} is i.i.d. generated according to $\mathcal{N}(0, 1)$. Then the new feature is $z = Ax$.

Code up KNN over the *hashed* vectors. Make sure you also hash vectors before prediction. Return the classification accuracy and runtimes for $m = 100, 1000, 10000$.

Ans. For this experiment, the accuracy and runtime numbers will vary, but the accuracy values should be in this ballpark.

Subsample hashing:

m	K	accuracy	runtime (sec)
100	1	0.5466	0.375
100	3	0.5189	0.365
100	5	0.5134	0.371
1000	1	0.7020	1.47
1000	3	0.7013	1.49
1000	5	0.7087	1.50
10000	1	0.8040	31.25
10000	3	0.8188	31.20
10000	5	0.8202	31.95

Random matrix hashing:

m	K	accuracy	runtime (sec)
100	1	0.6423	0.419
100	3	0.6067	0.426
100	5	0.5956	0.426
1000	1	0.8408	1.534
1000	3	0.8316	1.479
1000	5	0.8338	1.474
10000	1	0.9260	31.47
10000	3	0.9265	32.10
10000	5	0.9284	32.4

What are your thoughts on using hashing for KNN? In particular:

- Does the reduction in runtime justify the accuracy hit? In what scenarios would we prefer to use hashing vs not hashing?

  Ans. There is no standard answer to this question. It is just to exercise your reasoning skills. Clearly there is a performance tradeoff between accuracy and runtime, and often for large servers and important applications, the accuracy is important enough that we are ok with taking the runtime hit. (Or, translated to real world scenarios, we ask our boss for a bigger computer and because they have deep pockets we can easily get it.) But there may be many other scenarios where we don't really need perfect accuracy; in this case, hashing can get us to approximate answers with much less pain.

- Comment on the performance difference between hashing via subsampling and hashing via random matrices.

  Ans. In general, you should see random matrix hashing be at least slightly better in performance than subsample hashing. One intuitive reasoning for this is that the values to begin with are heavily quantized, so subsample hashing throws subspaces away, but maintains the low quantization in the remaining values. In contrast, random matrix hashing "spreads" the information into full 32 bit floating point precision, so even though subspaces are lost, not as much information is discarded. However, there is clearly also a runtime cost; in general, subsampling is an $O(n)$ procedure whereas matrix multiplication is $O(n \times n_{\text{small}})$ procedure. As with everything in life, there are tradeoffs.

Homework 3

1. **(1 pt.)** *Gradient descent for ridge regression.* Consider the problem

$$\underset{x}{\text{minimize}} \quad \overbrace{\underbrace{\frac{1}{2}\|Ax-b\|_2^2}_{=:f(x)} + \frac{\rho}{2}\|x\|_2^2}^{F(x)} \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$ and $n > m$. Justify all answers.

(a) Define $C = A^TA$. Recall that an eigenvalue of a symmetric matrix C is λ where $Cv = \lambda v$ for some vector v. Show that if $\lambda_{\max}^2$ is the largest eigenvalue of C and $\lambda_{\min}^2$ the minimum eigenvalue of C, then for any vector u,

$$\lambda_{\min}^2\|u\|_2 \le \|Cu\|_2 \le \lambda_{\max}^2\|u\|_2.$$

Hint: The *eigenvalue decomposition* of a symmetric matrix can be written as $C = V\Lambda V^T$ where $V = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$ contain the eigenvalues v_i of C, and $\Lambda = \mathbf{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n)$ contain the corresponding eigenvalues λ_i. (That is, $Cv_i = \lambda_i v_i$.) Under certain conditions which we will just assume [1] then V is an orthonormal matrix, e.g. $V^TV = VV^T = I$. Then, we can use this to form projections, e.g. pick any vector u. Then $VV^Tu = V^TVu = u$.

Ans. Pick any u. Then since $CV = VV^TCV = V\Lambda$,

$$Cu = CVV^Tu = V\Lambda V^Tu.$$

[1] eigenvalues must all have algebraic multiplicity = geometric multiplicity

Then

$$\|Cu\|_2^2 = u^TC^TCu = u^TV\Lambda V^TV\Lambda V^Tu = u^TV^T\Lambda\Lambda V^Tu = \sum_{i=1}^n \lambda_i^2(v_i^Tu)^2.$$

Well, in general,

$$\lambda_{\min}^2\sum_{i=1}^n(v_i^Tu)^2 \le \sum_{i=1}^n \lambda_i^2 v_i^Tu \le \lambda_{\max}^2\sum_{i=1}^n(v_i^Tu)^2$$

and $\sum_{i=1}^n(v_i^Tu)^2 = u^TV^TVu = u^Tu = \|u\|_2^2$.
Therefore,

$$\lambda_{\min}^2\|u\|_2^2 \le \|Cu\|_2^2 \le \lambda_{\max}^2\|u\|_2^2.$$

Taking the square roots everywhere gives us the result we want.

(b) Show that since $n > m$, then if $C = A^TA$ then $\lambda_{\min}(C) = 0$. Hint: The nonzero eigenvalues of AA^T and of A^TA are the same.

Ans. Without using more powerful tools like SVD and eigenvalue decompositions, the easiest way to argue this is that $AA^T \in \mathbb{R}^{m\times m}$ and has m eigenvalues, and $A^TA \in \mathbb{R}^{n\times n}$ and has n eigenvalues. Since $n > m$ and the nonzero eigenvalues of AA^T and A^TA are the same, it must be that at least one of the eigenvalues of A^TA is 0. To see that this is also the minimum eigenvalue, note that

$$A^TAu = \lambda u \iff u^TA^TAu = \lambda u^Tu \iff \lambda = \frac{\|Au\|_2^2}{\|u\|_2^2} \ge 0, \quad \forall \text{ eigenvectors } u.$$

In other words, *all eigenvalues of A^TA must be nonnegative.* Therefore, A^TA has a 0 eigenvalue and it is also the minimum eigenvalue.

(c) We say a function $f : \mathbb{R}^n \to \mathbb{R}$ is L-smooth if its gradient is L-Lipschitz, e.g. for some $L > 0$,

$$\|\nabla f(x) - \nabla f(y)\|_2 \le L\|x-y\|_2, \quad \forall x, y.$$

Is $f(x)$ as defined in (**??**) L-smooth? What about $F(x)$?

Ans. First, we need to compute some gradients:

$$\nabla f(x) = A^T(Ax-b), \quad \nabla F(x) = A^T(Ax-b) + \rho I.$$

Then

$$\|\nabla f(x) - \nabla f(y)\|_2 = \|A^TA(x-y)\|_2 \overset{\text{from part (a)}}{\le} \lambda_{\max}(A^TA)\|x-y\|_2$$

which shows that f is L-smooth, with $L = \lambda_{\max}(A^TA)$.
Note also that

$$\|\nabla F(x) - \nabla F(y)\|_2 = \|A^TA(x-y)\|_2 \le \lambda_{\max}(A^TA)\|x-y\|_2$$

which shows that F is also L-smooth, with $L = \lambda_{\max}(A^TA)$.

(d) We say a function $f : \mathbb{R}^n \to \mathbb{R}$ is μ-strongly convex if it is tangent to a quadratic function that is strictly under it; that is, for some $\mu > 0$,

$$f(x) - f(y) \ge \nabla f(y)^T(x-y) + \frac{\mu}{2}\|x-y\|_2^2, \quad \forall x, y.$$

If this is only true for $\mu = 0$, we say f is convex, but not strongly convex.

Is the function $f(x)$ in (**??**) μ-strongly convex? What about $F(x)$? Hint: This problem requires less work if you first answer for $F(x)$, and then take $\rho = 0$ for $f(x)$. **Ans.** First we do some messy calculations:

$$\begin{aligned}
F(x) - F(y) - \nabla F(y)^T(x-y) &= \frac{1}{2}\|Ax-b\|_2^2 + \frac{\rho}{2}\|x\|_2^2 - \frac{1}{2}\|Ay-b\|_2^2 + \frac{\rho}{2}\|y\|_2^2 - (A^T(Ay-b)+\rho y)^T(x-y) \\
&= \frac{1}{2}x^TA^TAx - b^TAx + \frac{1}{2}b^Tb + \frac{\rho}{2}x^Tx - \frac{1}{2}y^TA^TAy - b^TAy + \frac{1}{2}b^Tb - \frac{\rho}{2}y^Ty \\
&\quad -x^TA^TAy + x^TA^TAb - \rho y^Tx + y^TA^TAy - y^TA^Tb + \rho y^Ty \\
&\overset{\text{simplify}}{=} \frac{1}{2}x^TA^TAx + \frac{1}{2}y^TA^TAy - x^TA^TAy + \frac{\rho}{2}(x^Tx - 2x^Ty + y^Ty) \\
&= \frac{1}{2}\|Ax - Ay\|_2^2 + \frac{\rho}{2}\|x-y\|_2^2 \\
&\leq \frac{\lambda_{\min}^2(A) + \rho}{2}\|x-y\|_2^2.
\end{aligned}$$

So, $F(x)$ is μ-strongly convex with $\mu = \lambda_{\min}^2(A) + \rho$. But, from part (b), we know that $\lambda_{\min} = 0$, so more simply, $F(x)$ is just ρ-strongly convex, and $f(x)$ is not strongly convex.

(e) *Linear convergence.* We will now show that gradient descent on minimizing $F(x)$ converges *linearly*. First, recall that gradient descent iterates as

$$x^{(t+1)} = x^{(t)} - \alpha\nabla F(x^{(t)})$$

for some step size $\alpha > 0$.

i. Show that for any point x^* where $\nabla F(x^*) = 0$,

$$\|x^{(t+1)} - x^*\|_2 \leq c\|x^{(t)} - x^*\|_2$$

for some $c < 1$. What is c?
Ans. First, note that

$$\begin{aligned}
x^{(t+1)} &= x^{(t)} - \alpha A^T(Ax^{(t)} - b) - \alpha\rho x^{(t)} \\
&= (I - \alpha A^TA - \alpha\rho I)x^{(t)} + \alpha A^Tb.
\end{aligned}$$

Then, since

$$\nabla F(x^*) = 0 \iff A^TAx^* + \rho x^* = A^Tb$$

then

$$\begin{aligned}
x^{(t+1)} - x^* &= (I - \alpha A^TA - \alpha\rho I)x^{(t)} + \alpha A^Tb - x^* \\
&\overset{A^TAx^*+\rho x^* = A^Tb}{=} (I - \alpha A^TA - \alpha\rho I)x^{(t)} + \alpha(A^TAx^* + \rho x^*) - x^* \\
&= (I - \alpha A^TA - \alpha\rho I)(x^{(t)} - x^*).
\end{aligned}$$

This means $\|x^{(t+1)} - x^*\|_2 \leq c\|x^{(t)} - x^*\|_2$ for

$$c = \lambda_{\max}(I - \alpha A^TA - \alpha\rho I) = 1 - \alpha\rho - \alpha\lambda_{\min}(A^TA) = 1 - \alpha\rho.$$

ii. Use this to argue that gradient descent on (**??**) converges with *linear complexity*, e.g. the error $f(x^{(t)}) - f(x^*) = O(c^t)$. [2]
Ans. Now we can set up a recursion:

$$\|x^{(t)} - x^*\|_2 \leq c\|x^{(t-1)} - x^*\|_2 \leq c^2\|x^{(t-2)} - x^*\|_2 \leq \cdots \leq c^t\|x^{(0)} - x^*\|_2 = O(c^t).$$

[2] It's a weird convention, but we say $O(c^t)$ is a linear rate because the loglog graph looks linear. I don't make the rules, I just share them with you.

2. **(1pt)** *Linear regression without strong convexity still gets linear convergence.* Now consider linear regression with $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, and $\rho = 0$ (no ridge). That is, we consider only

$$f(x) = \frac{1}{2}\|Ax - b\|_2^2.$$

(a) *Linear algebra.* For a matrix A, the **nullspace** of A is the set

$$\mathbf{null}(A) = \{x : Ax = 0\},$$

and the **range** of A is the set

$$\mathbf{range}(A) = \{Ax \text{ for any } x\}.$$

Show that for

$$u = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \qquad v = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

then $u \in \mathbf{null}(A)$ and $v \in \mathbf{range}(A)$.

Ans. This can basically be done mechanically, by showing

$$Au = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1-1 \\ 1-1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

and picking $x = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$,

$$Ax = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} = v.$$

(b) *Linear decomposition theorem part 1.* Show that for *any* vector $u \in \mathbf{null}(A)$ and $v \in \mathbf{range}(A^T)$, it must be that $u^T v = 0$.

Ans. Since $v \in \mathbf{range}(A^T)$, it must be that $v = Ax$ for some x. Then

$$u^T v = u^T A^T x = (Au)^T x = 0.$$

(c) *Linear decomposition theorem part 2.* Argue also that for *any* vector x, there exist some $u \in \mathbf{null}(A)$ and $v \in \mathbf{range}(A^T)$ where $x = u + v$. Do this by providing two matrices P and Q where $Px = u$ and $Qx = v$, using $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. (This matrix will be unique.)

Ans. This is in fact true for *any* A, but for this problem we can see this by breaking down every $x = u + v$ as

$$v[1] = v[2] = \frac{1}{2}(x[1] + x[2])$$

and $u = x - v$. To show that $u \in \mathbf{null}(A)$,

$$Au = Ax - Av = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2}(x[1] - x[2]) \\ \frac{1}{2}(x[2] - x[1]) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

To show that $v \in \mathbf{range}(A^T)$, it suffices to say that the range of A are just vectors whose first and second component are the same. But, if we want to be super pedantic, we can just say

$$v = A^T \begin{bmatrix} \beta \\ 0 \end{bmatrix}$$

where $\beta = \frac{1}{2}(x[1] + x[2])$. In matrix form, we can write this as

$$P = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad Q = I - P = \frac{1}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

(d) *Linear regression doesn't pick up nullspace components.* Suppose $x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. Now we run the gradient descent method

$$x^{(t+1)} = x^{(t)} - \alpha \nabla f(x^{(t)}) \tag{2}$$

to get $x^{(1)}$, $x^{(2)}$, ...

Show that using this initial point, $Qx^{(t)} = 0$ for *all* t, where Q is the matrix computed in the previous question.

Ans. We use the hint to first argue that

$$\nabla f(x) = A^T \underbrace{(Ax - b)}_{z} = A^T z \in \textbf{range}(A^T).$$

Therefore, $\nabla f(x) = P\hat{u}$ for some $\hat{u}$, and

$$Q\nabla f(x) = \underbrace{QP}_{=\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}} \hat{u} = 0.$$

Therefore, for any t, if $Qx^{(t)} = 0$ then

$$Qx^{(t+1)} = Qx^{(t)} - \alpha Q\nabla f(x) = 0 - 0 = 0.$$

Since $Qx^{(0)} = 0$, by induction we have that $Qx^{(t)} = 0$ for all t.

(e) *Linear regression doesn't pick up nullspace components, another example.* Now suppose that for some $x^{(0)}$, $Qx^{(0)} = r \neq 0$. Again, we run the gradient descent method. Show that $Qx^{(t)} = r$ for all t. (That is, r does not depend on t!)

Ans. Following the same scheme as above, we can make the inductive argument that, if $Qx^{(t)} = r$, then

$$Qx^{(t+1)} = \underbrace{Qx^{(t)}}_{=r} - \alpha \underbrace{Q\nabla f(x^{(t)})}_{=0} = r.$$

Since $Qx^{(0)} = 0$, this completes the proof.

(f) Now consider a *reduced gradient descent problem*, where we minimize over a scalar variable v

$$\underset{v \in \mathbb{R}}{\text{minimize}}\ g(v) = \frac{1}{2}\|ASv - Pb\|_2^2, \quad S = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Argue that, using gradient descent

$$v^{(t+1)} = v^{(t)} - \alpha \nabla g(v^{(t)}),$$

the iterates $x^{(t)} = Sv^{(t)}$ are exactly those outputted by (**??**) where $x^{(0)} = Sv^{(0)}$.

Hint: Start by showing that $SS^T \nabla f(x) = \nabla f(x)$.

Ans. Starting with the hint, we have that $\nabla f(x) \in \textbf{range}(A)$, and therefore it must be that the two elements in $\nabla f(x)$ are the same. That is, $z = \nabla f(x)$, $z[1] = z[2]$. Then

$$SS^T z = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} z[1] \\ z[2] \end{bmatrix} = \frac{1}{2} \begin{bmatrix} z[1] + z[2] \\ z[1] + z[2] \end{bmatrix} = z.$$

Now suppose that $x^{(t)} = Sv^{(t)}$. Then, since by chain rule $\nabla g(v) = S^T \nabla f(Sv)$,

$$Sv^{(t+1)} = \underbrace{Sv^{(t)}}_{x^{(t)}} - \alpha S \underbrace{\nabla g(v^{(t)})}_{S^T \nabla f(x^{(t)})} = x^{(t)} - \alpha SS^T \nabla f(x^{(t)}) \overset{SS^T \nabla f(x) = \nabla f(x)}{=} x^{(t)} - \alpha \nabla f(x^{(t)}) = x^{(t+1)}.$$

That is, minimizing the reduced objective function $g(v)$ is *equivalent* to minimizing the original function $f(x)$!

(g) Finally, show that $g(v)$ is strongly convex in u.

Ans. This is easier if we start with $AS = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$, which means that

$$\begin{aligned} g(v) &= \frac{1}{2}\left((\frac{1}{\sqrt{2}}v - b[1])^2) + (\frac{1}{\sqrt{2}}v - b[2])^2) \right) \\ &= \frac{1}{2}\left(v^2 - 2v \cdot \underbrace{\frac{b[1]+b[2]}{\sqrt{2}}}_{\beta} + \underbrace{(b[1]^2 + b[2]^2)}_{\gamma} \right) \\ &= \frac{1}{2}v^2 - v\beta + 2\gamma. \end{aligned}$$

for some constant β. Then,

$$\begin{aligned} g(u) - g(v) - g'(v)(u-v) &= \frac{1}{2}(u^2 - v^2) - (u-v)\beta - (v-\beta)(u-v) \\ &\overset{\text{simplify}}{=} \frac{1}{2}(u-v)^2 \end{aligned}$$

which is μ-strongly convex with $\mu = 1$.

(h) Show that for this specific choice of A, gradient descent converges linearly, e.g. $f(x^{(t)}) - f(x^*) = O(c^t)$, and carefully state what c is.

Ans. In part (e), we showed that if F is ρ-strongly convex, then gradient descent converges linearly with $c = 1 - \alpha\rho$. Here, g is 1-strongly convex, so gradient descent on $g(v)$ converges at a linear rate with $c = 1 - \alpha$. To show this, we can simply write the recursion

$$(v^{(t+1)} - v^*) = (v^{(t)} - v^*) - \alpha g'(v^{(t)}) = (v^{(t)} - v^*) - \alpha(v^{(t)} - \beta) \overset{v^*-\beta=0}{=} (1-\alpha)(v^{(t)} - v^*).$$

Since gradient descent on g is equivalent to gradient descent on f, this shows that gradient descent on f is also converging at a linear rate, with the same c.

(i) Now consider *any* matrix A. Argue that this entire problem basically shows that gradient descent, minimizing linear regression, will *always* converge at a linear rate, and describe what c is.

Ans. This problem is a bit tough, and really requires some understanding of the big picture. What we have shown in this question is that gradient descent over linear regression doesn't seem to care about directions in the null space of A. So, we can always construct some function $g(v) = f(Sv)$ where $x = Sv + r$, $r \in \textbf{null}(A)$ and $Sv \in \textbf{range}(A^T)$. Then, running gradient descent over g will be equivalent to running gradient descent over f, but where g is strongly convex, with $\mu =$ the smallest nonzero eigenvalue of A^TA. For this value of μ, gradient descent will have convergence rate $O(c^t)$ for $c = 1 - \mu\alpha$.