Problem Set 1

Max Ruiz Luyten

Edited by Adityanarayanan Radhakrishnan

January 14, 2023

Linear Algebra

Problem 1 (T). Let $A \in \mathbb{R}^{n\times n}$ be a diagonalizable matrix with eigenvalues $\{\lambda_i\}_{i=1}^n$.

Let $A = Q\Lambda Q^{-1}$ and $\Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ given by the eigendecomposition.

(a) If $\{\lambda_i\}_{i=1}^n \subset (-1, 1]$, compute the eigenvalues of $(I + A)^{-1}$, where I is the identity matrix.

We have:

$$I + A = QQ^{-1} + Q\Lambda Q^{-1} = Q(I + \Lambda)Q^{-1} \ ;$$

where $(I + \Lambda) = diag(1 + \lambda_1, 1 + \lambda_2, \ldots, 1 + \lambda_n)$, $1 + \lambda_i \in (0, 2]$, $\forall i = 1, 2, \ldots, n$.
As Q, Q^{-1} and $I + \Lambda$ have inverses, we conclude:

$$(I + A)^{-1} = Q(I + \Lambda)^{-1}Q^{-1} \ \ ;$$

where $(I + \Lambda)^{-1} = diag(\frac{1}{1+\lambda_1}, \frac{1}{1+\lambda_2}, \ldots, \frac{1}{1+\lambda_n})$ are the corresponding eigenvalues.

(b) Compute the eigenvalues of A^k for fixed $k \in \mathbb{Z}_+$.

We prove by induction on k that $A^k = Q\Lambda^k Q^{-1}$. In particular, if λ_i is an eigenvalue of A, then λ^k is an eigenvalue of A^k, with the same multiplicity and with the same subspace of eigenvectors. The base case holds by definition for $k = 1$. For $k > 1$:

$$A^k = A^{k-1}A = Q\Lambda^{k-1}\not{Q}^{-1}(\not{Q}\Lambda Q^{-1}) = Q\Lambda^k Q^{-1} \ \ ;$$

where $\Lambda^k = diag(\lambda_1^k, \lambda_2^k, \ldots, \lambda_n^k)$ are the eigenvalues.

Note that in both (a) and (b) the eigenvectors for the matrices $(I - A)^{-1}$ and A^k are the same as they are for A.

Problem 2 (T).
(a) Let $\Sigma \in \mathbb{R}^{n\times n}$ be a diagonal matrix with diagonal entries $\{\sigma_i\}_{i=1}^n \in \mathbb{R}$. Compute $\Sigma^\dagger$.

Hint: Remember that σ_i can equal 0 for some i.
We know that the Moore-Penrose pseudo-inverse exists and is unique. Thus, one approach is to propose a candidate and check that it meets the requirements and uniqueness follows. We propose the matrix suggested in the lecture, $\tilde{\Sigma}$ where the entries are $\frac{1}{\sigma_i}$ for $\sigma_i \neq 0$ and $\sigma_i = 0$ otherwise. We next check that it satisfies the four conditions of the definition given in lecture 1. As the product of diagonal matrices is diagonal, we have that:

$$(\Sigma\tilde{\Sigma})^T = \Sigma\tilde{\Sigma} \ \ ; \ \ (\tilde{\Sigma}\Sigma)^T = \tilde{\Sigma}\Sigma.$$

Moreover, restricted to the subspace U generated by the eigenvectors that have a non-zero eigenvalue, $\Sigma\tilde{\Sigma}|_U = \tilde{\Sigma}\Sigma|_U = I|_U$, and $\ker(\Sigma) = \ker(\tilde{\Sigma})$, so that $\Sigma\tilde{\Sigma}\Sigma = \Sigma$ and $\tilde{\Sigma}\Sigma\tilde{\Sigma} = \tilde{\Sigma}$. Thus $\tilde{\Sigma} = \Sigma^\dagger$.

Alternatively, another solution is to find a diagonal matrix that satisfies the conditions of the pseudoinverse. Namely, weimpose the four conditions to a general matrix $\Sigma^\dagger = (b_{ij})_{i,j=0}^n$. We next rewrite each condition in terms of the matrix elements.

1. Condition 1 for a pseudoinverse reads $(\Sigma\Sigma^\dagger\Sigma)_{ij} = \sigma_i\sigma_j b_{ij} = (\Sigma)_{ij}$.
2. Condition 2 for a pseudoinverse reads $(\Sigma^\dagger\Sigma\Sigma^\dagger)_{ij} = \sum_{k=0}^n \sigma_k b_{ik} b_{kj} = b_{ij}$.
3. Condition 3 requires $(\Sigma\Sigma^\dagger)^T = \Sigma\Sigma^\dagger \iff \sigma_i b_{ij} = \sigma_j b_{ji}$.
4. Similarly, the last condition imposes $(\Sigma^\dagger\Sigma)^T = \Sigma^\dagger\Sigma \iff \sigma_i b_{ji} = \sigma_j b_{ij}$.

The last two conditions require that for every pair $i \neq j$ such that $\sigma_i = 0, \sigma_j \neq 0$, we have $b_{ij} = b_{ji} = 0$. On the other hand, the first condition implies that if $i \neq j$ and $\sigma_i, \sigma_j \neq 0$, then $b_{ij} = b_{ji} = 0$. Hence, we have that given any i, either $\sigma_i = 0$ or for all $j \neq i$, $b_{ij} = 0$.

Therefore, for $i \neq j$, in the second condition, all terms in the sum are 0 and thus $b_{ij} = 0, \forall i \neq j$. Similarly, for $i = j$ there is only one non-zero term, thus imposing $\sigma_i b_{ii}^2 = b_{ii}$, so if $\sigma_i = 0$, then $b_{ii} = 0$, and if $\sigma_i \neq 0$, then $b_{ii} = \sigma_i^{-1}$.

(b) Let $A = U\Sigma V^T \in \mathbb{R}^{n\times n}$ by the Singular Value Decomposition (SVD). Verify that $A^\dagger = V\Sigma^\dagger U^T$.
Hint: Check that $V\Sigma^\dagger U^T$ satisfies the properties of the pseudoinverse from Definition 6 of Lecture 1.

1. $AA^\dagger A = U\Sigma\cancel{V^T}\cancel{V}\Sigma^\dagger\cancel{U^T}\cancel{U}\Sigma V^T = U\Sigma\Sigma^\dagger\Sigma V^T = U\Sigma V^T = A$, where we have used the first property of the pseudoinverse for Σ.
2. $A^\dagger AA^\dagger = V\Sigma^\dagger\cancel{U^T}\cancel{U}\Sigma\cancel{V^T}\cancel{V}\Sigma^\dagger U^T = V\Sigma^\dagger\Sigma\Sigma^\dagger U^T = V\Sigma^\dagger U^T = A^\dagger$, where we have used the second property of the pseudoinverse for Σ.
3. $(AA^\dagger)^T = (U\Sigma\Sigma^\dagger U^T)^T = U(\Sigma\Sigma^\dagger)^T U^T = U\Sigma\Sigma^\dagger U^T = AA^\dagger$, where we have used the third property of the pseudoinverse for Σ.
4. $(A^\dagger A)^T = (V\Sigma^\dagger\Sigma V^T)^T = V(\Sigma^\dagger\Sigma)^T V^T = V\Sigma^\dagger\Sigma V^T = A^\dagger A$, where we have used the forth property of the pseudoinverse for Σ.

Problem 3 (C, Required). Generate a square matrix $A \in \mathbb{R}^{5000\times5000}$ and a vector $b \in \mathbb{R}^{5000\times1}$ with entries drawn i.i.d from a standard normal distribution. Compare the runtime of solving $Ax = b$ using the numpy `solve` function against that of first computing A^{-1} with the numpy `inv` function and then computing $x = A^{-1}b$.
Remark: For timing, one can use the `time.time()` function in Python. This exercise is to reinforce that the `solve` function should be used in place of `inv` when solving square matrix linear systems.
`numpy.linalg.solve` uses the LU decomposition in the general case, so that the solving complexity is $O(n^2)$ to decompose and solve the two triangular systems. `numpy.linalg.inv` also uses the LU decomposition, but it solves n system of equations, one for each canonical vector e_i, so that the solving complexity is $O(n^2)$ to decompose and $O(n^3)$ to solve the $2n$ triangular systems, plus the $O(n^2)$ matrix-vector product.
Using the following sample code:

```
import numpy as np
import time

# Sample matrices
A = np.random.normal(size=(5000,5000))
b = np.random.normal(size=5000)

t1 = time.time()
x1 = np.linalg.solve(A,b) # Linalg solve
t2 = time.time()
```

```
x2 = np.linalg.inv(A)@b # Inverse solve
t3 = time.time()

# Verify similarity of solutions
assert(np.isclose(x1, x2, 0, 1e-10).all()), "Incorrectly solved system"

# Output times
print("Time spent by solve:", t2-t1)
print("Time spent by inv:", t3-t2)
```

Listing 1: Code for problem 1.3

With a laptop, we got times of 0.69s and 2.37s using `solve` and `inv`, respectively.

Problem 4 (C). Generate a square matrix $A \in \mathbb{R}^{100\times 100}$ with entries drawn i.i.d. from a standard normal distribution. Compute U, Σ, V^T via the numpy `svd` function. Compute $A^\dagger$ via the numpy `pinv` function and verify that it matches $V\Sigma^\dagger U^T$.

We provide the code below.

```
import numpy as np
import time

n, m = 100, 100

A = np.random.normal(size=(n,m))

# Numpy pseudoinverse
t1 = time.time()
Ainv1 = np.linalg.pinv(A)
t2 = time.time()

# Hand written pseudoinverse
u, s, vh = np.linalg.svd(A) # SVD

sinv = np.zeros((m, n), dtype=complex)      # Allocate \Sigma inv
s[np.where(s == 0)[0]] = np.inf             # 0 where s == 0 when taking 1./s
sinv[:min(m,n), :min(m,n)] = np.diag(1./s)  # The :min is not needed if m=n
Ainv2 = vh.transpose()@sinv@u.transpose()   # SVD for the inverse
t3 = time.time()

print(np.linalg.norm(Ainv1-Ainv2))  # of the order of 1e-15

# The time both methods take is very similar
print("pinv time: ", t2-t1)
print("Hand pinv time: ", t3-t2)
```

Listing 2: Code for problem 1.4

Analysis

Problem 5 (T, Required). For $A \in \mathbb{R}^{m\times n}$ and $x, b \in \mathbb{R}^m$, let $\mathcal{L}(x) = \|Ax - b\|_2^2$. Compute $\frac{\partial \mathcal{L}}{\partial x}$.
Hint: Try computing the partial derivatives $\frac{\partial L}{\partial x_i}$ and group the terms into a vector. Are there any obvious patterns that emerge?
Remarks: This is a required exercise since this computation will appear in Lecture 2.
Let's write $(A\mathbf{x})_i = \sum_{j=1}^n a_{ij}x_j$. Then $(A\mathbf{x}-\mathbf{b})_i = -b_i + \sum_{j=1}^n a_{ij}x_j$, so that $\mathcal{L}(\mathbf{x}) = \sum_{i=1}^n \left(-b_i + \sum_{j=1}^n a_{ij}x_j\right)^2$.
We can now easily differentiate the loss with respect to any variable x_k by using the linearity of the derivative

and the chain rule.

$$\begin{aligned}\frac{\partial \mathcal{L}(x)}{\partial x_k} &= \sum_{i=1}^{n} 2\left(-b_i + \sum_{j=1}^{n} a_{ij}x_j\right)\frac{\partial}{\partial x_k}\left(\sum_{j=1}^{n} a_{ij}x_j\right) \\ &= \sum_{i=1}^{n} 2\left(-b_i + \sum_{j=1}^{n} a_{ij}x_j\right)a_{ik}\end{aligned}$$

All in all,

$$\begin{aligned}\left(\frac{\partial \mathcal{L}(x)}{\partial x}\right)_k &= 2\sum_{i=1}^{n}(Ax-b)_i\, a_{ik} \\ &= 2\left[(Ax-b)^T A\right]_k\end{aligned}$$

As the gradient is a vector in $\mathbb{R}^n$, it corresponds to the transpose of the matrix inside the brackets, $\nabla\mathcal{L}(x) = 2A^TAx - 2A^Tb$

Problem 6 (T, Required). In Lecture 1, we briefly introduced the notion of norms on function spaces. In particular, we presented the example of the L^2 norm of a function $f : \mathbb{R} \to \mathbb{R}$, which in general is given by:

$$\|f\|_{L^2}^2 = \int_{-\infty}^{\infty} |f(x)|^2 dx$$

In this course, we will commonly use another norm, which we call the $L^2(\mu)$ norm, that is defined as follows. For $f : \mathbb{R} \to \mathbb{R}$:

$$\|f\|_{L^2(\mu)}^2 = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} |f(x)|^2 e^{-\frac{x^2}{2}} dx$$

Compute $\|f\|_{L^2(\mu)}$ when $f(x) = \max(x, 0)$ and $\|g\|_{L^2(\mu)}$ when $g(x) = \begin{cases} 1 & \text{if } x \geq 0. \\ 0 & \text{if } x < 0. \end{cases}$.

Hint: Recall, that the probability density function of the standard normal Gaussian is given by:

$$p(x) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{x^2}{2}\right)$$

Use the fact that the integral of a density must be 1 to compute $\|g\|_{L^2(\mu)}$. Similarly, use the expectation of the Gaussian to compute $\|f\|_{L^2(\mu)}$.

Remarks. The computations in this problem will re-appear in the derivation of normalizing factors for the NNGP and NTK later in this course.

Note that

$$\begin{aligned}||f||_{L^2(\mu)}^2 &= \frac{1}{\sqrt{2\pi}}\left(0 + \int_0^{\infty} |x|^2 e^{-\frac{x^2}{2}} dx\right) \\ &= \frac{1}{2}\left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} x^2 e^{-\frac{x^2}{2}} dx\right) \\ &= \frac{1}{2}\end{aligned}$$

where in the second equality we have used that the integrand is even and the last equality is a consequence of the fact that if $X \sim N(0, 1)$ then $\mathbb{E}[X^2] = 1$. Hence, $||f||_{L^2(\mu)} = \frac{1}{\sqrt{2}}$. Similarly, for the step function g we

have

$$\begin{aligned} ||g||^2_{L^2(\mu)} &= \frac{1}{\sqrt{2\pi}}\left(0 + \int_0^\infty e^{-\frac{x^2}{2}}dx\right) \\ &= \frac{1}{2}\left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^\infty e^{-\frac{x^2}{2}}dx\right) \\ &= \frac{1}{2} \end{aligned}$$

where for the last equality we use that a $P(\Omega) = 1$ if $(\Omega, \mathcal{F}, P)$ is a probability space, in this case with P defined through the gaussian pdf on $\Omega = \mathbb{R}$. Hence, $||g||_{L^2(\mu)} = \frac{1}{\sqrt{2}}$.

Problem 7 (C). Generate vectors $a, b \in \mathbb{R}^{500}$ with entries drawn i.i.d standard normal distribution.
(a) Compute $\|a\|_2^2$ via the numpy `norm` function and via your own implementation. Check that the two match.
(b) Compute the mean squared error (MSE) between vectors a, b given by $\frac{1}{500}\|a - b\|_2^2$.

Note that $a - b \sim N(0, 2I)$, which implies $\frac{1}{500}||a-b||_2^2 = \frac{1}{500}\sum_{i=0}^{500} 2(\frac{a-b}{\sqrt{2}})_i^2 \sim \frac{2}{500}\chi^2(500)$.

The mean $\mathbb{E}[\frac{2}{500}\chi^2(500)] = 2$ and the 90% confidence interval is $[1.80, 2.21]$.

```python
import numpy as np

n = 500

# Sample the two vectors
a = np.random.normal(size=n)
b = np.random.normal(size=n)

# Hand crafted norm as a sum of squares
aNorm2 = np.sum(np.square(a))
bNorm2 = np.sum(np.square(b))

# Difference between hand crafted norm and linalg norm ~1e-14
print("||a||^2 error: ", abs(aNorm2 - np.linalg.norm(a, ord=2)**2))
print("||b||^2 error: ", abs(bNorm2 - np.linalg.norm(b, ord=2)**2))

# MSE ~ 2
print("MSE: ", np.linalg.norm(a-b, ord=2)**2/n)
```

Listing 3: Code for problem 1.7

Probability

Problem 8 (T, Required). Compute the following integral:

$$\int_{-\infty}^\infty e^{-x^2+4x}dx$$

Hint: This problem is in the probability section for a reason. Complete the square and see if a known integral appears.
Remarks: This technique will be useful in computing the NNGP and NTK later on in this course.

Completing squares, $-x^2 + 4x = -(x^2 - 4x + 4) + 4 = (x-2)^2 + 4$. The remaining computations follow from the hint.

$$\int_{-\infty}^\infty e^{-x^2-4x}dx = \sqrt{\pi}e^4\frac{1}{\sqrt{2\pi\frac{1}{2}}}\int_{-\infty}^\infty e^{-\frac{(x-2)^2}{2\frac{1}{2}}}dx = \sqrt{\pi}e^4$$

Problem 9 (T). Let $v, w \overset{i.i.d.}{\sim} \mathcal{N}(\mathbf{0}, I_{n\times n})$. Note $v, w \in \mathbb{R}^n$.
(a) Compute $\mathbb{E}_w[ww^T]$ and $\mathbb{E}_w[\|w\|_2^2]$.
(b) Compute $\mathbb{E}_{(w,v)}[\langle v, w\rangle]$, where $\langle\cdot,\cdot\rangle$ is the standard dot product on vectors in $\mathbb{R}^n$.

Recall that if $\{\omega_i\}_{i=1}^n$ are iid standard normal gaussian variables $\omega_i \sim N(0,1)$, we have

$$\mathbb{E}[\omega_i\omega_j] = \begin{cases} \mathbb{E}[\omega_i^2] = 1 \text{ if } i = j \\ \mathbb{E}[\omega_i]\mathbb{E}[\omega_j] = 0 \text{ if } i \neq j \end{cases}$$

Therefore,

$$\mathbb{E}[\omega\omega^T] = \begin{pmatrix} \mathbb{E}[\omega_1\omega_1] & \cdots & \mathbb{E}[\omega_1\omega_n] \\ \vdots & \ddots & \vdots \\ \mathbb{E}[\omega_n\omega_1] & \cdots & \mathbb{E}[\omega_n\omega_n] \end{pmatrix} = I_{n\times n}$$

On the other hand, $\mathbb{E}[||\omega||_2^2] = \mathbb{E}[\sum_{i=0}^n \omega_i^2] = \sum_{i=0}^n \mathbb{E}[\omega_i^2] = n$.

Lastly, if v, w are also standard gaussian and their entries are chosen independently we have

$$\mathbb{E}[\langle v, w\rangle] = \mathbb{E}[\sum_{i=0}^n v_i w_i] = \sum_{i=0}^n \mathbb{E}[v_i w_i] = \sum_{i=0}^n \mathbb{E}[v_i]\mathbb{E}[w_i] = 0$$

Problem 10 (C). Generate $A \in \mathbb{R}^{n\times n}$ with entries drawn from an i.i.d. standard normal distribution.
(a) For $n \in \{10, 100, 1000\}$, compute $\frac{1}{n}AA^T$. What matrix does $\frac{1}{n}AA^T$ approach for large n?
(b) Fix $n = 10$. Sample m such matrices $\{A_i\}_{i=1}^m$ for $m \in \{10, 100, 1000\}$ and compute $\frac{1}{m}\sum_{i=1}^m A_i^2$. To what matrix does this sum converge?

```python
import numpy as np

# (a)
nvals = (10, 100, 1000)
for n in nvals:
    # Sample matrix and print 1/n* (A @ A^T)
    a = np.random.normal(size=(n, n))
    aaT = a@a.T/n
    print("1/n AA^T for n = {}: ".format(n), aaT)

    # Verify that all elements are close to the identity
    idErr = aaT - np.eye(n)
    print(max(-idErr.min(), idErr.max())) # Max absolute value of the array

# (b)
mvals = (10, 100, 1000)
n = 10
for m in mvals:
    # Sample m matrices and print avgSq = 1/m * SUM(A*A)
    ma = np.random.normal(size = (m, n, n))

    avgSq = np.zeros((n,n))
    for i in range(m):
        avgSq += np.linalg.matrix_power(ma[i, :, :], 2)
    avgSq /= m
    print("Sum A^2 for n = {}, m = {}: ".format(n, m), avgSq)

    # Verify that all elements are close to the identity
    idErr = avgSq - np.eye(n)
    print(max(-idErr.min(), idErr.max())) # Max absolute value of the array
```

Listing 4: Code for problem 1.10

Note that in both cases the resulting matrix seems to converge to the identity. This can be verified theoretically by using the law of large numbers, which ensures that if $X, \{X_i\}_{i=1}^n$ are iid, then $\frac{1}{n}\sum_{i=1}^n X_i$ converges almost surely to $\mathbb{E}[X]$. We then have that the elements in the diagonal of $\frac{1}{n}AA^T$ are $(\frac{1}{n}AA^T)_{ii} = \frac{1}{n}\sum k = 0^n a_{ik}^2 \to \mathbb{E}[X^2] = 1$ for $X \sim N(0,1)$. For the non diagonal terms $i \neq j$ we have $(\frac{1}{n}AA^T)_{ij} = \frac{1}{n}\sum k = 0^n a_{ik}a_{jk} \to \mathbb{E}[X_1X_2] = 0$ for independent $X_1, X_2 \sim N(0,1)$.

Similarly $\left(\frac{1}{m}\sum_{i=1}^m A_i^2\right)_{jk} = \frac{1}{m}\sum_{i=1}^m \sum_{l=1}^n a_{jl}^{(i)} a_{lk}^{(i)}$. Thus, the sum converges to $\sum_{l=1}^n \mathbb{E}[a_{jl}^{(i)} a_{lk}^{(i)}]$. If $j \neq k$ then $a_{jl}^{(i)}, a_{lk}^{(i)}$ are independent for all l, and thus the non diagonal terms converge to 0. For the diagonal terms we have that for $l = j = k$ the expectation is $\mathbb{E}[(a_{jj}^{(i)})^2] = 1$ and all the other terms are products of independent standard gaussians and so their expectation is zero.

The maximum absolute values of $AA^T - I$ for $n = 10, 100, 1000$ are $0.62, 0.42, 0.17$ on a given run.

For $\sum A^2$ with $n = 10$ and $m = (10, 100, 1000)$ the maximum absolute values of $\sum A^2 - I$ are $2.55, 0.84, 0.26$.