(1) Here are some easy formulas for calculating the pseudo-inverse for a matrix of maximal rank (the generic case). Let \( M \) be a non-zero matrix with a singular value decomposition \( M = UDV^H \).

(a) Show that \( M \) is injective if and only if \( V \) is square and invertible. Conclude in this case that \( M^H M \) is invertible and \( M^+ = (M^HM)^{-1}M^H \).

**Proposition 1.** \( M \) is injective if and only if its singular value decomposition \( M = UDV^H \) has a \( V \) that is square and invertible. In this case, \( M^H M \) is invertible and \( M^+ = (M^HM)^{-1}M^H \).

**Proof.** Let \( M \) be an \( r \times c \) matrix. Suppose that \( M \) is injective, so that \( \text{rank}(M) = c \) because the kernel is zero. Then \( D \) is a \( c \times c \) matrix and so \( V^H \) is also \( c \times c \). \( V^H \) must already be injective (lest \( M \) not be injective), and since it is also onto as part of a singular value decomposition, it is bijective thus invertible (also true because it’s an operator). \( V \) is of course also square and has an inverse.

Now suppose that \( V \) is square an has an inverse. Then \( V^H \) is also square and has an inverse, and is \( c \times c \). Thus \( D \) is also \( c \times c \) and so the rank of \( M \) is \( c \). The dimension of the image is the same as the domain, so \( M \) must be injective.

Thus in this case \( M^H M = (UDV^H)^H UDV^H = VDV^H UDV^H = VD^H DV^H \) is really the product of four invertible square matrices and so is invertible itself. We see that
\[
M^+ = VD^{-1}U^H \\
= (VD^{-1}(D^H)^{-1}V^H)(VD^H U^H) \\
= (VD^H DV^H)^{-1}M^H \\
= (M^HM)^{-1}M^H
\]

\( \square \)

(b) Show that \( M \) is surjective if and only if \( U \) is square and invertible. Conclude in this case that \( MM^H \) is invertible and \( M^+ = M^H (MM^H)^{-1} \).

**Proposition 2.** If \( r \times c \) matrix \( M \) is surjective if and only if \( U \) in its singular value decomposition is square and invertible. So \( MM^H \) is invertible and \( M^+ = M^H (MM^H)^{-1} \).
Proof. If $M$ is surjective, then $\text{rank}(M) = r$, and so $D$ is $r \times r$ and square, and then so is $U$. $U$ is surjective because $M$ is and so as an operator it must be invertible.

If we suppose that $U$ is square and invertible then it must be $r \times r$. $D$ must also be $r \times r$ and so the rank of $M$ is $r$ and thus $M$ is surjective. Thus in this case $MM^H = UDV^H (UDV^H)^H = UDV^H (VD^H U^H) = \sqrt{D} \sqrt{D} U^H$ is really the product of four invertible square matrices and so is invertible itself. We see that

$$M^+ = VD^{-1}U^H$$
$$= (VD^H U^H)(U(D^H)^{-1}D^{-1}U^H)$$
$$= (VD^H U^H)(UDD^H U^H)^{-1}$$
$$= M^H (MM^H)^{-1}$$

\[ \square \]

(2) The **trace** of a matrix, denoted $\text{Tr}(M)$, is the sum of the diagonal elements of the matrix:

$$\text{Tr}(M) = \sum_i M[i, i]$$

This simple formula, which ignores almost all elements in the matrix, is surprisingly useful (for example in Galois theory and group representations).

(a) If the products $NM$ and $MN$ are both defined for matrices $M$ and $N$, prove $\text{Tr}(NM) = \text{Tr}(MN)$.

**Proposition 3.** If the products $NM$ and $MN$ are both defined for matrices $M$ and $N$, then $\text{Tr}(NM) = \text{Tr}(MN)$.

**Proof.** If both $NM$ and $MN$ are defined, then the matrices are both square, so let’s say they are $n \times n$. Then

$$\text{Tr}(NM) = \sum_{k=1}^{n} \sum_{j=1}^{n} N[k, j]M[j, k]$$

and

$$\text{Tr}(MN) = \sum_{k=1}^{n} \sum_{j=1}^{n} M[k, j]N[j, k].$$

If you change the order of the summations and the multiplication in one of these expressions, you get the other, so they are the same! \[ \square \]

3/3

(b) Let $T : V \to V$ be a linear operator on a finite dimensional vector space. Define $\text{Tr}(T) = \text{Tr}(\mathcal{M}_B^V(T))$ for a basis $B$ of $V$. Show that $\text{Tr}(T)$ is independent of the choice of $B$. (Hint: If $B$ and $B'$ are bases of $V$, show that there exists an invertible matrix $N$ such that $\mathcal{M}^B_B(T) = N^{-1} \mathcal{M}^B_{B'}(T)N$.) Thus the trace can be defined for any linear operator.

**Proposition 4.** The trace of a linear operator $T : V \to V$ can be well-defined by setting $\text{Tr}(T) = \text{Tr}(\mathcal{M}_B^V(T))$. 

Proof. We must show that \( \text{Tr}(T) \) is independent of the choice of \( B \). First, we let \( N = \mathcal{M}_B^B(I) \), so that \( N \) is invertible with \( N^{-1} = \mathcal{M}_B^{B'}(I) \). Thus \( \mathcal{M}_B^B(T) = N^{-1} \mathcal{M}_B^{B'}(T) N \). If we remember all these matrices are squares of the same size, then we then have

\[
\text{Tr} (\mathcal{M}_B^B(T)) = \text{Tr} \left( N^{-1} \left( \mathcal{M}_B^{B'}(T) N \right) \right) \\
= \text{Tr} \left( \left( \mathcal{M}_B^{B'}(T) N \right) N^{-1} \right) \\
= \text{Tr} \left( \mathcal{M}_B^{B'}(T) \right) .
\]

\( \square \)

3/3 Response: Good use of formalism for matrix of an operator.

(c) The Frobenius norm of a matrix, denoted \( \| M \|_F \), is the norm of the matrix as a vector:

\[
\| M \|_F = \sqrt{\sum_{i,j} |M[i,j]|^2}
\]

Prove: \( \text{Tr}(M^H M) = \| M \|_F^2 \).

**Proposition 5.** \( \text{Tr}(M^H M) = \| M \|_F^2 \).

**Proof.** This really only requires applying the definitions:

\[
\text{Tr}(M^H M) = \sum_{k=1}^{n} \sum_{j=1}^{n} M^H[k,j]M[j,k] \\
= \sum_{k=1}^{n} \sum_{j=1}^{n} M[j,k]M[j,k] \\
= \sum_{j,k} |M[i,j]|^2 \\
= \| M \|_F^2.
\]

\( \square \)

(d) For any non-zero matrix \( M \), show that \( \| M \|_F^2 \) is the sum of the squared singular values of \( M \).

**Proposition 6.** For any non-zero matrix \( M \), \( \| M \|_F^2 \) is the sum of the squared singular values of \( M \).

**Proof.** Since \( \text{Tr}(M^H M) = \| M \|_F^2 \), I just need to show that \( \text{Tr}(M^H M) \) is the sum of the squared singular values of \( M \). If we do a singular value decomposition to make \( M = U D V^H \), as shown earlier, we get \( M^H M = V D^H D V^H \). Add columns full of zeros to the right side of \( V \) calling it \( V' \), and to the right and bottom of \( D \) calling it \( D' \) so that all are square matrices of the same size. Then we get that

\[
\text{Tr}(V D^H D V^H) = \text{Tr} \left( \left( V' D' \right)^H \left( D' V' \right) \right) = \text{Tr} \left( \left( D V^H \right) \left( V' D' \right) \right) = \text{Tr}(D' D^H),
\]

because \( V' V' \) has an identity matrix the size of \( D \) in the
upper left corner. The diagonal elements of $D^H D'$ are precisely the squared singular values of $M$, so we are done. □

Response: You don’t have to add any zeros. $\text{Tr}(M^HM) = \text{Tr}(VD^2V^H) = \text{Tr}(D^2V^HV) = \text{Tr}(D^2)$. 3/3

(3) One of the important uses of singular value decomposition is approximating large matrices by smaller ones. In this problem you will create a graphic image with 10,000 data values, then approximate it surprising well using only 200 data values. Throughout this problem only real matrices are used, so we will use the transpose symbol instead of the Hermitian symbol.

(a) Suppose a real matrix has a real singular value decomposition $M = UDV^T$. We’ll assume that the matrix is $r \times c$ with rank $n$. What are the sizes of $U$, $D$ and $V$? What is the total number of real numbers in these matrices (only count the diagonal elements of $D$). How many real numbers are in $M$? If the rank of $M$ is much smaller than the size of $M$, you can save memory by storing $U$, $D$ and $V$ instead of storing $M$. $U$ is $r \times n$, $D$ is $n \times n$, and $V$ is $c \times n$, so the number of real numbers in these matrices is $rn + n + cn = n(r + c + 1)$. The number of real numbers in $M$ is $rc$. 3/3

(b) The $i^{th}$ singular value approximation of $M$ is constructed from the $i$ largest singular values as follows: Let $U_i$ be the first $i$ columns of $U$. Define $V_i$ similarly, and let $D_i$ be the upper-left $i \times i$ corner of $D$. $D_i$ is a diagonal matrix with the $i$ largest singular values of $M$. Show that $U_i$ and $V_i$ are orthogonal. Let $M_i = U_i D_i V_i^T$. Show that $M_i$ is the same size as $M$ and has rank $i$. $M_i$ is a rank $i$ approximation of $M$. For what value of $i$ does $M_i = M$?

**Proposition 7.** $U_i$ and $V_i$ are orthogonal.

*Proof.* $U$ and $V$ are orthogonal because their columns are orthonormal sets. Any subset of an orthonormal set is also orthonormal, and so $U_i$ and $V_i$ which are just the first $i$ columns of $U$ and $V$ are also orthogonal. □

**Proposition 8.** $M_i$ is the same size as $M$ and has rank $i$.

*Proof.* Because $U_i$ is $r \times i$, $D_i$ is $i \times i$, and $V_i$ is $c \times i$, the product $M_i = U_i D_i V_i^T$ is $r \times c$ , the same size as $M$. The ranks of $U_i$, $D_i$, and $V_i$ (and thus $V_i^T$) are all $i$ because all have linearly independent columns. And so the rank of $M$ is the minimum of all these, which is again $i$. □

3/3 **Response:** Be careful that you aren’t just saying that the rank of a product is equal to the minimum rank of the factors. The product rank can be less.

**Proposition 9.** If $i = n$ then $M = M_i$.

*Proof.* If $i = n$, then all the matrices that compose $M$ and $M_i$ are the same, so then $M = M_i$. □
(c) You are going to use Octave to create a graph of the surface \( z = \frac{\sin(r)}{r} \), where \( r \) is the polar coordinate in the \( x,y \)-plane. Most of the commands that follow are terminated by a semicolon. The semicolon suppresses output, which is desired when creating large matrices.

\[
\begin{align*}
x &= y = \text{linspace}(-8,8,100); \\
[mx,my] &= \text{meshgrid}(x,y); \\
r &= \sqrt{mx.^2 + my.^2}; \\
z &= \sin(r) ./ r; \\
\text{mesh}(x,y,z)
\end{align*}
\]

To find the size of the matrix \( z \), enter

\[
\text{size}(z)
\]

\[
\text{octave:2> size}(z)
\]
\[
\text{ans} =
\]
\[
100 \\ 100
\]

How big is \( z \)? How many real numbers are stored in \( z \)?

\( z \) is \( 100 \times 100 \) and has \textbf{10,000 real numbers}.

Let’s create another graph of the same function, this time using only about 200 values.

\[
\begin{align*}
xx &= yy = \text{linspace}(-8,8,14); \\
[mx,my] &= \text{meshgrid}(x,y); \\
r &= \sqrt{mx.^2 + my.^2}; \\
zz &= \sin(r) ./ r; \\
\text{mesh}(xx,yy,zz)
\end{align*}
\]
How well does the new graph approximate the original graph? **Good**

What is the size of \( zz \)? \( 14 \times 14 \) How many real numbers are stored in \( zz \)? **196**

Now you will use the singular value decomposition to create another approximation using about the same number of data values. You will create the first singular value decomposition of \( z \).

\[
[u, d, v] = \text{svd}(z);
\]

\[
u_1 = u(:,1);
v_1 = v(:,1);
d_1 = d(1,1);
z_1 = u_1*d_1*v_1';
\]

\[
\text{mesh}(x, y, z_1)
\]
How well does this graph approximate the original graph? **Wow!** That’s amazing! I did it twice just to make sure it wasn’t the same graph. You will notice though, that it has lost its ring-like nature. This is due to the fact that the new graph shows the results of a large two-dimensional multiplication table with the entries of the first columns of $V$ and $U$ written along the $x$ and $y$ axes (scaled by $D$). You can see the hills (positive numbers) created by positive times positive or negative times negative, and you can see the vallies where the results are negative instead. What are the sizes of $u_1, v_1, d_1$ and $z_1$. They are $100 \times 1, 1 \times 1, 1 \times 100,$ and $100 \times 100,$ respectively. If you stored $u_1, v_1$ and $d_1$ instead of storing $z_1$, how many real numbers would you have to store? **I would have to store 201 numbers instead of 10,000.** You have created a very good replica of the original graph, and you have reduced the data storage by what percent? **It was reduced by 97.99%**

(d) Extra credit. There is something going on here that I (Dr. Meredith) don’t understand. The reason that the rank 1 approximation $z_1$ to $z$ is so good is that $z$ itself is a low rank matrix. What is rank($z$)? **Octave says it’s 8, though I suspect much of that may be due to round off error.** The matrix $z$ is a $100 \times 100$ matrix holding the values $\sin(\sqrt{x^2 + y^2})/\sqrt{x^2 + y^2}$ as $x$ and $y$ run over 100 values between $-8$ and $8$. Why does this matrix have such low rank? The last graph above shows just what you can get with 100 scalar multiplies of the same vector of length 100. If you think about it, it shouldn’t be too surprising that in some ways it’s a great approximation. I actually don’t think, though that we only need 8 vectors and all their linear combinations to recreate $z$ exactly. In fact, decreasing the tolerance setting for Octave’s rank() function I get interesting results:

```
octave:129> rank(z,1e-14)
ans = 8
octave:130> rank(z,1e-15)
ans = 11
octave:131> rank(z,1e-16)
ans = 83
octave:132> rank(z,1e-17)
ans = 96
octave:133> rank(z,1e-18)
ans = 98
```

Octave is actually using 64-bit float numbers here, and so that’s why at a certain point the results are obviously false because we know the rank is at most 50 for reasons of symmetry.

When I found out that Octave’s rank() function uses Octave’s singular value decomposition routines for its calculations, I
decided to try to let it reduce the matrix to reduced row echelon form instead:

octave:133> rank(rref(z,1e-14))
anst = 17
octave:134> rank(rref(z,1e-15))
anst = 27
octave:135> rank(rref(z,1e-16))
anst = 95

In fact, to get a rank of 8 this way, I need to increase the tolerance to 1e-9!

Another observation is that if spread the values out for x and y between 300 and -300 instead of 8 and -8, but still use 100 data points, then Octave’s rank() function returns 50 as we would expect.

octave:145> x = y = linspace(-300,300,100);
[mx,my] = meshgrid(x,y);
r = sqrt(mx.^2 + my.^2); z = sin(r)./r; rank(z)
anst = 50

So what is happening here? The smoothness of the graph when zoomed in is playing tricks on Octave’s rank() routine. Perhaps the most reliable calculation above returned a rank of 27 (out of 50). How much of the other 23 are due to rounding error and how much are due to mathematical coincidence, I’m not sure. I’m fairly confident, though, that since none of our boundaries have anything to do with $\pi$, that the rank really is 50.

Last, just observe what happens when I combine some of the columns of z:

In fact, if I take the rank of the first 8 columns we see that the rank() function already falters whereas using rank(rref()) does not:

octave:164> rank(z(1:8,:))
anst = 7

octave:171> rank(rref(z(1:8,:)))
anst = 8

Take a look as some of these differences:

octave:152> z(:,52) - z(:,51)
anst = 
  -1.1059e-04  
  -4.8445e-05  
  2.0358e-05  
  9.4651e-05  
  1.7304e-04  
  2.5391e-04  
  3.3547e-04  
  4.1571e-04
4.9252e-04
5.6363e-04
6.2667e-04
6.7925e-04
...

octave:151> z(:,52) - 4*z(:,51) +6*z(:,50)-4*z(:,49)+z(:,48)
ans =
-3.5358e-06
-4.0523e-06
-4.5245e-06
-4.9346e-06
-5.2640e-06
-5.4932e-06
-5.6024e-06
-5.5713e-06
-5.3797e-06
-5.0076e-06
-4.4358e-06

This is my suspicion: the second partial derivatives of this function are relatively small, so that for very small changes in $x$ (or $y$) the change in $z$ is fairly constant, and fairly small since $z$ is small as well. So locally, a lot of this looks linear, linear enough to fool Octave’s 64-bit floating point resolution. I think we should compute the rank of this matrix using a system that has arbitrary precision before trying to prove that the rank of the matrix is really so low.

3/3 Response: Interesting exploration. I’ll post it for the class to read.