Homework 2 : Md Mahin (1900421)

1 Answers

1. (9 points) Determine the (a) eigenvalues, (b) determinant, and (c) singular values of a Householder reflector. For the eigenvalues, give a geometric argument as well as an algebric proof.

   Answer:
   (a):We know, Householder reflector $F = I - 2\frac{vv^*}{v^*v}$
   We also know, the relation between eigenvector v (which on the span of the reflector) and eigenvalue λ for any matrix F(which is also the reflector matrix) is:
   $Hv = \lambda v$
   or, $v - 2\frac{vv^*}{v^*v}v = \lambda v$
   or, $v - 2v = \lambda v$
   or, $\lambda = -1$
   So, for any vector within the span of the reflector, the eigenvalue is -1 algebraically.
   The geometric representation of this will be, any vector from the span of the reflector, if is multiplied by the reflector, will be on the same plane, but the direction will be different. Again, lets take another vector u, which is perpendicular to v. u represents vector from remaining $n-1$ dimension. So, the relation will be: $Hu = \lambda u$
   or, $u - 2\frac{vv^*}{v^*v}u = \lambda u$
   as, $v^*u = 0$ because of the orthogonality, or, $u - 0 = \lambda u$
   or, $\lambda = 1$
   So, for any orthogonal dimension to the span of the reflector, the eigenvalue is 1 algebraically.
   Geometrically, we can say that any vector perpendicular to the vector v, it is span of F, it will be geometrically invariant.

(b)Here if we create the orthonormal basis for the q_i vectors of the reflector, where $i = 1, 2, 3...n$, q_{n-1} vectors will be orthogonal to the vector u. So, considering eigunvalue -1 for the vector v from the span and 1 for rest u vectors, our determinant will be multiplication of all. It is,
$det(F) = -1 * 1^{n-1}$
so, $det(F) = -1$

(c)We can easily prove, householder reflector is symmetric/hermatian considering it's normalized general form, it is,
$F = I - qq^*$
here, $F^* = (I - qq^*)^*$
or, $F^* = I^* - (q^*)^* q^*$
or, $F^* = I - qq^*$
so, $F^* = F$

For, any symmetric matrix, singular values are absolute eigenvalues, for our reflector that is 1 and -1. So, all the singular values $\sigma_i = 1$ for $i = 1, 2...n$.

2. (3 points)(a) Write a MATLAB function [W,R] = house(A) that computes an implicit representation of a full QR factorization $A = QR$ of an $m \times n$ matrix A with $m \geqslant n$ using Householder reflections. The output variables are a lower-triangle matrix $W \in \mathbb{C}^{m \times n}$ whose coloumns are the vector v_k defining the successive Householder reflections, and a triangular matrix $R \in \mathbb{C}^{n,n}$.

(3 points) (b) Write a MATLAB function $Q = formQ(W)$ that takes the matrix W produced by house as input and generates a corresponsing $m \times n$ orthogonal matrix Q. Make a 4×3 matrix A and generate the Q using your house() and formQ(). Compare with Matlab qr() and make comments.
Answer:
(a) The function is implemented in code file Problem_2_a_b.m under the name [W,R] = house(A).
(b) The function is implemented in code file Problem_2_a_b.m under the name Q=formQ(W).

Comments: The Q, R matrix generated by house() and formQ() function are visually exactly similar to the Q1 and R1 generated by matlab qr() function. On the other hand, when I computed $||Q - Q1||$ and $||R - R1||$, I got results 1.0689e-15 and 4.6184e-15 respectively. Both are near machine epsilon indicating very little errors between the matrices.

3. (4 points) Let Z be the matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 7 \\ 4 & 2 & 3 \\ 4 & 2 & 4 \end{pmatrix}$$

Compute three reduced QR factorization of Z in MATLAB: by the Gram-Schmidit routine mgs of Homework1(Question 13), by the Householder routines house and formQ of Question 2, and by MATLAB's built-in command [Q,R] = qr(Z,0). Compare these three and comment on any differences you see.

Answer:

(a) The solution is implemented in code file Problem_3.m.

Comments: If we compare the Q and R generated by three methods, House holder R is similar to R generated by matlab qr() function. They differ with Gram-Scimidt not in value but in sign. Similar situation can be seen for Q's generated by all three functions. Another difference is householder Q is more elaborate having more columns.

When I calculated stability of the three algorithms, e.g. when I calculated $||Z - (Q * R)||$ and $||R - R1||$ for modified Gram-Schimdit, Householder and matlab qr I got following results respectively 8.8818e-16, 4.6738e-15, 5.7308e-15. All three are near machine epsilon, however Gram-Schimdit seems performs best having least error. Second is housholder and third performance is from matlab QR.

4. (16 points) Take $m = 50, n = 12$. Using MATLAB's linsapce, define t to be the m-vector corresponding to linearly spaced grid points from 0 to 1. Using MATLAB's vander and fliplr, define A to be the $m \times n$ matrix associated with least square fitting on this grid by a polynomial

of degree $n-1$. Take b to be the function $cos(4t)$ evaluated on the grid. Now, calculate and print (to sixteen-digit precision) the least squares coefficient vector x by six methods:
(a) Formation and solution of the normal equations, using MATLAB's \
(b) QR factorization computed by mgs(modified Gram-Schimidt, Homework1 -Question 13))
(c) QR factorization computed by house (Householder triangularization, Question 2)
(d) QR factorization computed by MATLAB's qr (also Householder triangularization)
(e) $x = A\backslash b$ in MATLAB (also based on QR factorization)
(f) SVD, using MATLAB's svd
(g) The calculations above will produce six lists of twelve coefficients. In each list, mark out the digits that appear to be wrong(affected by rounding error). Comment on what differences you observe. Do the normal equations exhibit instability? You do not have to explain your observations.

Answer:
(a) The solution is implemented in code file Problem_4_a_b_c_d_e_f_g.m.
(b) The solution is implemented in code file Problem_4_a_b_c_d_e_f_g.m.
(c) The solution is implemented in code file Problem_4_a_b_c_d_e_f_g.m.
(d) The solution is implemented in code file Problem_4_a_b_c_d_e_f_g.m.
(e) The solution is implemented in code file Problem_4_a_b_c_d_e_f_g.m.
(f) The solution is implemented in code file Problem_4_a_b_c_d_e_f_g.m.

-0.999999996011848	1.000000000304192	1.000000000996588	1.000000000996605	1.000000000996606	1.000000000996608
0.000000919141483	-0.000000207865170	-0.000000422741829	-0.000000422743252	-0.000000422743296	-0.000000422743245
-8.000029609461993	-7.999989477104218	-7.999981235712919	-7.999981235678678	-7.999981235677416	-7.999981235678847
0.000371278591804	-0.000195407206918	-0.000318762923310	-0.000318763322135	-0.000318763333747	-0.000318763320215
10.664290761688340	10.668474341389546	10.669430793897623	10.669430796514671	10.669430796563514	10.669430796500022
0.008832847330653	-0.009457796041637	-0.013820280001311	-0.013820290513418	-0.013820290604798	-0.013820290443356
-5.709982955709934	-5.659557264779021	-5.647075647883502	-5.647075620768057	-5.647075620776509	-5.647075620983563
0.037790245032587	-0.052277389334820	-0.075315989673511	-0.075316035544790	-0.075316035157460	-0.075316035111808
1.562141000557293	1.666197785085729	1.693606925294266	1.693606975959675	1.693606975155598	1.693606975396566
0.101389981691293	0.026331163058959	0.006032135249895	0.006032100041100	0.006032100841448	0.006032100496409
-0.413485535389099	-0.382753404849379	-0.374241713949964	-0.374241699970661	-0.374241700377927	-0.374241700178140
0.095037452729814	0.089584038385429	0.088040577886213	0.088040575467180	0.088040575552226	0.088040575507807

(g) The values are added in the above figure. Here the first column represents results from a, second column results from b , third column results from c, , fourth column results from d , fifth column results from e and sixth column is results from f. Those are seems too wrong are marked in red. Red marked. The red marked ones show huge distance from other values and sign change. On the other hand the Yalow marked ones are having slightly grater distance than others. It seems the results from c,d,e,f are very close. The results from b is slightly

distant from others, but most of the results from a are wrong. So we can say method from number a is more unstable than others.

5. (6 points) The floating point system **F** defined by $x = \pm(m/\beta^t)\beta^c$ includes many integers, but not all of them.
(a)Give an exact formula for the smallest positive integer n that does not belong to **F**.
(b)In particular, what are the values of n for IEEE single and double precision arithmetic?
(c)Figure out a way to verify this result for your own computer. Specifically, design and run a program that produces evidence that $n-3, n-2$ and $n-1$ belong to **F** but n does not. What about $n+1, n+2$, and $n+3$? Please NOTE: For more information on formula $x = \pm(m/\beta^t)\beta^c$ click here.

Answer:
(a)Here given,
$x = \pm(m/\beta^t)\beta^c$
With this formula, the highest integer digit that is possible is β^t, which will need exactly t precision digit. However, to represent the digit n=$\beta^t + 1$ we need $t + 1$ digits where most and least significant once are non zero. So, the value n=$\beta^t + 1$ is not possible with the above number system and it is the smallest positive integer that is not possible.

(b)In IEEE single precision arithmetic, $\beta = 2$ and $t = 24$,
so, $n = 2^{24} + 1$
In IEEE double precision arithmetic, $\beta = 2$ and $t = 53$,
so, $n = 2^{53} + 1$

(c)The solution is implemented in code file Problem_5_c.m.
On the code I have used 24 bit for single precision, created $n1 = 2^{24} - 3; n2 = 2^{24} - 2; n3 = 2^{24} - 1; n4 = 2^{24}; n5 = 2^{24} + 1; n6 = 2^{24} + 2; n7 = 2^{24} + 3$. Also taken 53 bit for double precision and and created $n1 = 2^{53} - 3; n2 = 2^{53} - 2; n3 = 2^{53} - 1; n4 = 2^{53}; n5 = 2^{53} + 1; n6 = 2^{53} + 2; n7 = 2^{53} + 3$. Later to verify a number exists or not, I have subtracted it from it's adjacent number. If the difference is 1 number exist, if it is 0 or 2 it does not exist. For every $n-3$,$n-2$,$n-1$ I got 1 as the difference and for n the difference is zero, for $n + 1$,

$n+2$,$n+3$ the difference is two. So we can say, $n-3$,$n-2$,$n-1$ exist and n,$n+1$, $n+2$,$n+3$ does not exist. [Proved]

6. (3 points)(a) Show that $(1+O(\epsilon_{machine}))(1+O(\epsilon_{machine})) = 1+O(\epsilon_{machine})$. The precise meaning of this statement is that if f is a function satifying $f(\epsilon_{machine}) = (1+O(\epsilon_{machine}))(1+O(\epsilon_{machine}))$ as $\epsilon_{machine} \to 0$, then f also satifies $f(\epsilon_{machine}) = 1+O(\epsilon_{machine})$ as $\epsilon_{machine} \to 0$.
(3 points) (b) Show that $(1+O(\epsilon_{machine}))^{-1} = 1+O(\epsilon_{machine})$.

Answer:
(a)L.H.S=$(1+O(\epsilon_{machine}))(1+O(\epsilon_{machine}))$
$=1+2O(\epsilon_{machine})) + O(\epsilon_{machine}))^2$
Here, $O(\epsilon_{machine}))^2$ or $O(\epsilon^2_{machine}))$ will be very small and thus negligible.
If we consider, $2O(\epsilon_{machine}))$ as $O(\epsilon_{machine}))$ ignoring the scalar 2, than we can write,
L.H.S= $1+O(\epsilon_{machine})$
So, L.H.S = R.H.S (proved)

(b)We know, $(1-x)^{-1} = 1-x+x^2-x^3-...$
So, $(1+O(\epsilon_{machine}))^{-1} = 1-O(\epsilon_{machine})+O(\epsilon_{machine})^2-O(\epsilon_{machine})^3+....$
or, $(1+O(\epsilon_{machine}))^{-1} = 1+O(\epsilon_{machine})^2 + O(\epsilon_{machine})^4 + ..-O(\epsilon_{machine}) + O(\epsilon_{machine})^3 + ...$
as, the power of $O(\epsilon_{machine})$ always produces very smaller result, we can ignore them. So the final result will be equivalent to,
$(1+O(\epsilon_{machine}))^{-1} = 1-O(\epsilon_{machine})$
We can write this as,
$(1+O(\epsilon_{machine}))^{-1} = 1+O(\epsilon_{machine})(proved)$

7. (16 points) Each of the following problems describes an algorithm implemented on a computer satifying the axioms (13.5) and (13.7). For each one, state whether the algorithm is *backward stable, stable but not backward stable, or unstable*, and prove it or at least give a reasonably convincing argument. Be sure to follow the definitions as given in the text.
(a) Data: $x \in \mathbb{C}$. Solution: $2x$, computed as $x \oplus x$.
(b) Data: $x \in \mathbb{C}$. Solution: x^2, computed as $x \otimes x$.
(c) Data: $x \in \mathbb{C}\backslash\{0\}$. Solution: 1, computed as $x \oslash x$. (A machine satisfying (13.6) will give exactly the right answer, but for our definitions

are based on the weaker conditions (13.7).)
(d) Data: $x \in \mathbb{C}$. Solution: 0, computed as $x \ominus x$. (Again, a real machine may do better than our definitions based on (13.7).)
(e)Data. none. Solution: e, computed by summing $\sum_{k=0}^{\infty} 1/k!$ from left to right using $\otimes$ and $\oplus$, stopping when a summand is reached of magnitude $< \epsilon_{machine}$.
(f) Data: none. Solution: e, computed by the same algorithm as the above except with the series assumed from right to left.
(g) Data: none. Solution: π, computed by doing an exhaustive search to find the smallest floating point number x in the interval [3,4] such that $s(x) \otimes s(x') \leqslant 0$. Here $s(x)$ is an algorithm that calculates $sin(x)$ stably in the given interval, and x' denotes the next floating point number after x in the floating point system.

Answer:
(a)Here, given, for any input x we will get the result $2x$. For floating point system, we know, $fl(x) = x(1 + \epsilon_1)$.
Now, $\tilde{f}(x) = fl(x) \oplus fl(x)$
or, $= (x(1 + \epsilon_1) + x(1 + \epsilon_1))(1 + \epsilon_2)$
or,=2x$(1+\epsilon_1)(1 + \epsilon_2)$
or, $= 2x(1 + \epsilon_3)$
or, $= f(\tilde{x})$
Here, $\tilde{x} = x(1 + \epsilon_3)$or, $= (x(1 + \epsilon_1) - x(1 + \epsilon_1))(1 + \epsilon_2)$

The difference between x and $\tilde{x}$ depends on the value of x. We can say, at least for some x, $\tilde{f}(x)$ and $f(\tilde{x})$ will be very close.
Now, to prove the backward stability, we need to check if is solving a nearby problem.
So, $\frac{|\tilde{x}-x|}{|x|} = \frac{x(1+\epsilon_3)-x}{x} = |(1 + \epsilon_3)| = O(\epsilon_{machine})$
So, we can say it is solving a nearby problem and thus it is backward stable. Moreover, as it is backward stable, it is stable.(proved)

(b)Here, given, for any input x we will get the result x^2. For floating point system, we know, $fl(x) = x(1 + \epsilon_1)$.
Now, $\tilde{f}(x) = fl(x) \otimes fl(x)$
or, $= (x(1 + \epsilon_1) \times x(1 + \epsilon_1))(1 + \epsilon_2)$
or,=(x$(1+\epsilon_1))^2(1 + \epsilon_2)$
or, $= x^2(1 + \epsilon_1)^2(1 + \epsilon_2)$
or, $= (x\sqrt{(1 + \epsilon_1)}(1 + \epsilon_2))^2$

or, $= f(\tilde{x})$
Here, $\tilde{x} = x\sqrt{(1+\epsilon_1)}(1+\epsilon_2)$, where the term $\sqrt{(1+\epsilon_1)}$ will be very small. The difference between x and $\tilde{x}$ depends on the value of x. We can say, at least for some x, $\tilde{f}(x)$ and $f(\tilde{x})$ will be very close.
Now, to prove the backward stability, we need to check if is solving a nearby problem.
So, $\frac{|\tilde{x}-x|}{|x|} = \frac{x\sqrt{(1+\epsilon_1)}(1+\epsilon_2)}{x} = |\sqrt{(1+\epsilon_1)}(1+\epsilon_2)| = O(\epsilon_{machine})$
So, we can say it is solving a nearby problem and thus it is backward stable. Moreover, as it is backward stable, it is stable. (proved)

(c)Here, given, for any input x we will get the result 1. For floating point system, we know, $fl(x) = x(1+\epsilon_1)$.
Now, $\tilde{f}(x) = fl(x) \oslash fl(x)$
or, $= (x(1+\epsilon_1) \div x(1+\epsilon_1))(1+\epsilon_2)$
or,$=(1+\epsilon_2)$
or, $= f(\tilde{x})$
Here, for any $\tilde{x}$ we will get $f(\tilde{x}) = (1+\epsilon_2)$ and the result is independent of any x and also not exactly same as $\tilde{f}(x)$. So, if we want to calculate any perturbation to the input, it is, if we want to calculate,
$\frac{|\tilde{x}-x|}{|x|}$
we do not find any $\tilde{x}$ for which $\tilde{f}(x) = f(\tilde{x})$
we can say the problem is not backward stable.
Now, to prove the stability, we need to check if the outputs are close or not.
So, $\frac{|\tilde{f}(x)-f(\tilde{x})|}{|f(\tilde{x})|} = \frac{1+\epsilon_2-1}{1} = |\epsilon_2| = O(\epsilon_{machine})$
So, we can say the problem is stable. Moreover, we can say it is not backward stable, but it is stable. (proved)

(d)Here, given, for any input x we will get the result 0. For floating point system, we know, $fl(x) = x(1+\epsilon_1)$.
Now, $\tilde{f}(x) = fl(x) \ominus fl(x)$
or, $= (x(1+\epsilon_1) - x(1+\epsilon_1))(1+\epsilon_2)$
or,$=0(1+\epsilon_2)$
or, $= 0$
or, $= f(\tilde{x})$
Here, $\tilde{x} = 0$. The difference between x and $\tilde{x}$ depends on the value of x. We can say, for some x, $\tilde{f}(x)$ and $f(\tilde{x})$ will be very close or same.

Now, to prove the backward stability, we need to check if is solving a nearby problem.
So, $\frac{|\tilde{x}-x|}{|x|} = \frac{x(1+\epsilon_1)-x}{x} = |(\epsilon_1)| = O(\epsilon_{machine})$
So, we can say it is solving a nearby problem and thus it is backward stable. Moreover, as it is backward stable, it is stable.(proved)

(e) There is no input data. For backward stability we need to have the input data and need to be able to make some perturbation to the input. So, we can say it is at least not backward stable. So, all we need to test the forward stability. However this method cannot be forward stable as the errors will decrease slower than $O(\epsilon_{machine})$.
More explicitly, our problems seems like following,
$1 \oplus \frac{1}{2} \oplus \frac{1}{6} \oplus ....$
So, the error terms will be,
$((1 + \frac{1}{2})(1 + \epsilon_1) + \frac{1}{6})(1 + \epsilon_2)....$
So, for the second term the increase of error overall will be drop down to $O(\epsilon^2)$. Considering on the n^{th} term the coefficient ϵ_k will converge to e. Here the n satisfies $n \cong \frac{1}{\epsilon_{machine}}$ and the error will be bounded by $\cong n\epsilon_{machine}$. So the error will increase at least as fast as $\cong n\epsilon_{machine}$ as the n will grow slowly while decreasing $\epsilon_{machine}$ and it will not be $O_{machine}$. This is why it is not forward stable.

(f)Similar to e as there is no input data, we can say it is at least not backward stable. So, all we need to check is forward stability. Here there will be n terms where n is a slowly growing function of $\frac{1}{\epsilon_{machine}}$satisfying $n! \cong \frac{1}{\epsilon_{machine}}$ similar to problem number e. But as the function is now decreasing from right to left, the error does not grow with n. Now our problem becomes something like this:
$\frac{1}{n!} \oplus \frac{1}{(n-1)!} \oplus \frac{1}{(n-2)!} \oplus .... \oplus 1$
So the error will look like,
$((\frac{1}{n!} + \frac{1}{(n-1)!})(1 + \epsilon_{n-1}) + \frac{1}{(n-2)!})(1 + \epsilon_{n-2})...$
So, the error ϵ_k will be bounded by,
$\sum_{k=1}^{n-1} \epsilon_k \sum_{j=k}^{n} \frac{1}{j!} \leqslant \epsilon_m \sum_{k=1}^{n-1} \sum_{j=k}^{n} \frac{1}{j!}$
$= \left[\frac{n-1}{n!} + \sum_{j=k}^{n} \frac{j}{j!}\right] \cong \epsilon_m e = O(\epsilon_m).$
So, we can say, instead of getting closer to e as like left to right procedure from before, here, with k there will be small growth of the coefficients of ϵ_k and it will converge. So the overall error will be bound to $O(\epsilon_m)$. So the problem is forward stable.

(g)Similar to e and f as there is no input data, we can say it is at least not backward stable. So all we need is to check the forward stability. Here, considering the calculation of $sin(x)$ is stable, so with a stable perturbation, $(x + \delta)$, we have
$\tilde{sin}(x) = sin(x + \delta).[1 + O(\epsilon_m)]$ for any $\delta = |x|O(\epsilon_m))$.
It means, for any $x = \pi$, $\tilde{sin}$ function will change it's sign only when $|\delta| = \pi O(\epsilon_m))$. So we can say, $\tilde{sin}(x) \otimes sin(x') \leqslant 0$, where x' is a floating point successor of x and it go along with $\pi|1 + O(\epsilon_m)|$. So it is forward stable. (proved)

8. (9 points) Consider an algorithm for the problem of computing the (full) SVD of a matrix. The data for this problem is a matrix A, and the solution is three matrices U (unitary), Σ (diagnol), and V (unitary) such that A $= U\Sigma V^*$. (We are speaking here of explicit matrices U and V, not implicit representtions as products of reflectors.)
(a) Explain what it would mean for this algorithm to be backward stable.
(b) In fact, for a simple reason, this algorithm cannot be backward stable. Explain.
(c) Fortunately, the standard algorithms for computing the SVD are stable. Explain what stability means for such an algorithm.

Answer: (a) Backward stability of SVD means, for slight perturbation to the input, that is, $\tilde{A} = A + \sigma A$, that will generate another SVD, $\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^*$ and $\frac{||\sigma A||}{||A||} = O(\epsilon_m)$

(b) The algorithm cannot be backward stable as the change in dimension with input to output is huge. The output matrices has dimension of $n^2 + m^2 + n$, if we just consider just the diagonal calculation of Σ. On the other hand, the input has the dimension of $m \times n$. As a result, slight perturbation in the input will cause much wider change to the output. As a result, we will have the matrices $\tilde{U}$ and $\tilde{U}$ as approximation of unitary matrices, they will not be fully unitary. So, the algorithm cannot be backward stable.

(c) To be stable, SVD algorithms needs to maintain following property- after a slight perturbation $\tilde{A} = A + \sigma A$, the relative error between the SVD $\tilde{U}\tilde{\Sigma}\tilde{V}^*$ of matrix A and the SVD of $A + \sigma A$ will be order of $O(\epsilon_m achine)$. It is:

$\frac{||\tilde{U}\tilde{\Sigma}\tilde{V}^* - (A+\sigma A)||}{||(A+\sigma A)||} = O(\epsilon_m achine)$
Or, $\frac{||\sigma A||}{||A||} = O(\epsilon_m achine)$

9. (9 points) The idea of this exercise is to carry out an experiment analogous to the one described in this lecture, but for the SVD instead of QR factroization.
(a) Write a MATLAB program that constructs a 50×50 matrix $A = U * S * V'$, where U and V are random orthogonal matrices and S is a diagnol matrix whose diagnol entries are random uniformly distributed numbers in [0,1], sorted into non increasing order. Have your program compute $[U2, S2, V2] = svd(A)$ and the norms of $U - U2$, $V - V2$, $S - S2$, and $A - U2 * S2 * V2'$. Do this for five matrices A and comment on the result. (Hint: Plots of $diag(U2' * U)$ and $diag(V2' * V)$ may be informative)
(b) Fix the signs in your computed SVD so that the difficulties of (a) go away. Run the program again for five random matrices and comment on the various norms. Do they have a connection with cond(A)?
(c) Replace the diagonal entries of S by their sixth powers and repeat (b). Do you see significant differences between the results of this exercise and those of the experiment for QR factorization?

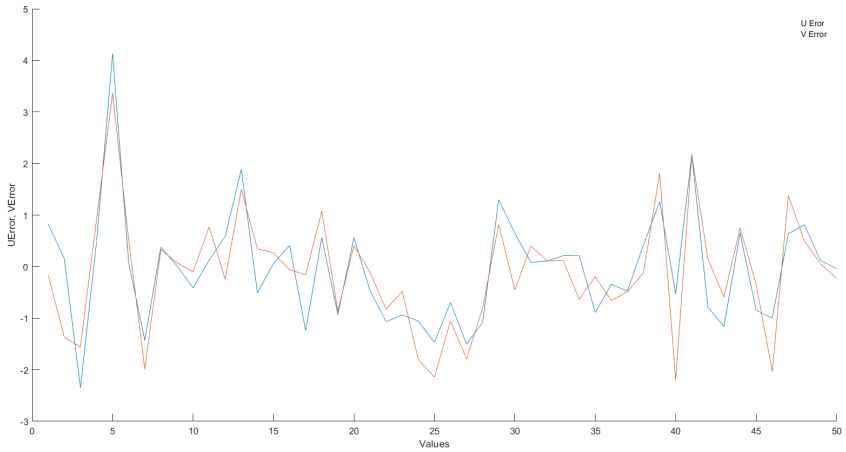
Answer:
(a)Table 1 represents the results I got from the analysis of 9(a)

$\|\|U2 - U\|\|$	$\|\|V2 - V\|\|$	$\|\|S2 - S\|\|$	$\|\|A - U * S * V'\|\|$
13.6706	13.8471	85.5121	1.5301e-13
13.7230	13.7258	73.0080	1.8607e-13
14.4385	13.5500	79.4003	1.5133e-13
14.2038	13.6025	86.2141	3.1511e-13
13.6316	14.5304	73.8300	1.8658e-13

Table 1: .

From the data we can see that, for U2,U; V2,V, and S2,S the norm of difference is high, indicating high dissimilarity among the those matrices. However, the two generated matrices A and $A2 = U * S * V'$, the norm of difference is very low and we can say we are at least solving some nearby problem. We can see the same situation by observing

Plots of $diag(U2' * U)$, $diag(V2' * V)$ given below.



(b) To change sign I have considered column by column, e.g. if $U2(j)' * U(j) < 0$, than I have changed the sign of both $U2(j)$ and $V2(j)$.

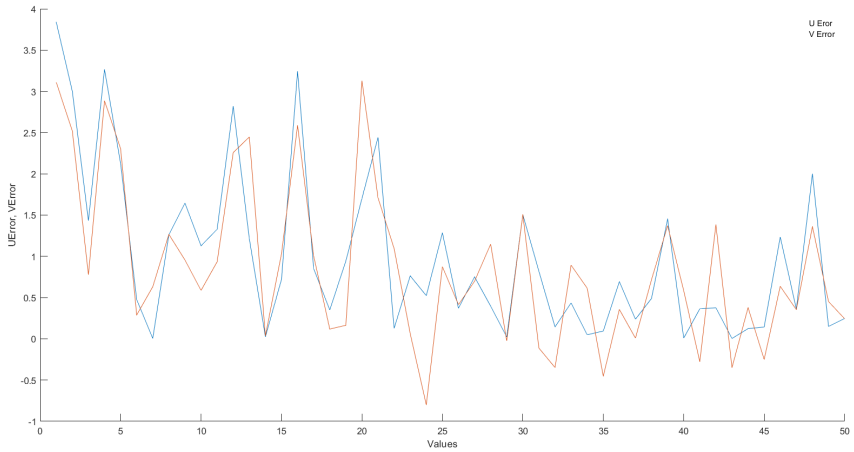
Table 2 represents the results I got from the analysis of 9(b)

$\|\|U2 - U\|\|$	$\|\|V2 - V\|\|$	$\|\|S2 - S\|\|$	$\|\|A - U * S * V'\|\|$	cond(A)
14.8750	12.9285	77.8067	3.5890e-13	1.8083e+04
13.8978	14.0016	77.7169	4.1190e-13	7.5314e+04
13.8653	13.9317	91.3190	3.0297e-13	3.0577e+04
13.6811	14.2503	80.4798	1.9838e-13	2.4834e+05
13.4053	13.5969	73.7121	4.1294e-13	6.9780e+06

Table 2: .

From the data we can see that, even after changing the sign the results are similar. For U2,U; V2,V, and S2,S the norm of difference is high, indicating high dissimilarity among the those matrices and the two generated matrices A and $A2 = U * S * V'$, the norm of difference is very low and we can say we are at least solving some nearby problem. I think there is a relation between cond(A) with the problem, as in all these cases the condition is very high.

We can see the same situation by observing Plots of $diag(U2' * U)$, $diag(V2' * V)$ given below.



(c)

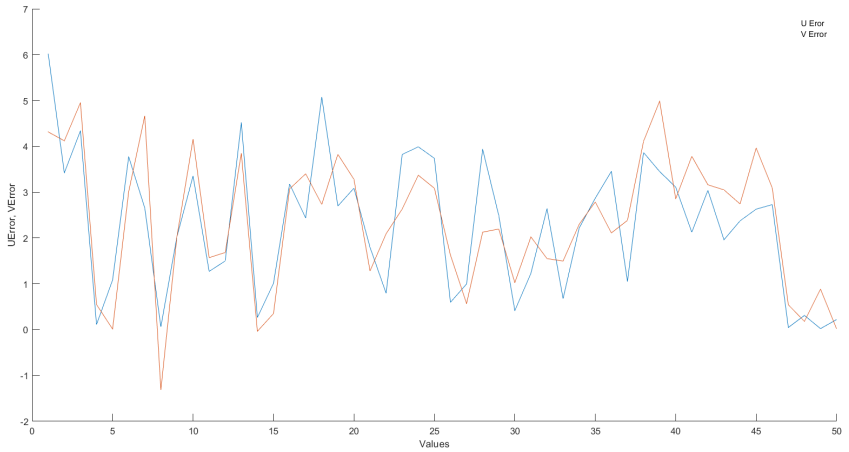
Table 3 represents the results I got from the analysis of 9(b)

$\|\|U2-U\|\|$	$\|\|V2-V\|\|$	$\|\|S2-S\|\|$	$\|\|A-U*S*V'\|\|$	cond(A)
13.0280	13.8558	41.2717	1.2409e-13	4.3281e+14
13.4377	12.6315	57.7881	1.2799e-13	9.4813e+11
13.4654	12.9029	73.5674	7.7760e-14	3.7855e+17
13.6125	13.5706	45.8382	1.3274e-13	6.1492e+11
13.1600	13.3854	48.5122	1.1046e-13	2.5781e+13

Table 3: .

After changing the magnitude of diagonal to the power of 6, the norm distance for all improved. However the condition number got worse.

We can see the same situation by observing Plots of $diag(U2' * U)$, $diag(V2' * V)$ given below.



10. (6 points) Suppose an $m \times m$ matrix A is written in the block form $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$, where A_{11} is $n \times n$ and A_{22} is $(m-n) \times (m-n)$. Assume that A satisfies the condition - **for each k with $1 \leqslant k \leqslant m$, the upper-left $k \times k$ block $A_{1:k,1:k}$ is nonsingular.**
(a) Verify the formula

$$\begin{pmatrix} I & \\ -A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}$$

for elimination of the block A_{21}. The matrix $A_{22} - A_{21}A_{11}^{-1}A_{12}$ is known as *Schur complement* of A_{11} in A.
(b) Suppose A_{21} is eliminated row by row by means of n steps of Gaussian elimination. Show that the bottom-right $(m-n) \times (m-n)$ block of the result is again $A_{22} - A_{21}A_{11}^{-1}A_{12}$.

Answer:
(a) Let's multiply

$$\begin{pmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{pmatrix}$$

with the matrix

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

and we get:

$$\begin{pmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

$$= \begin{pmatrix} A_{11} & A_{12} \\ -A_{21}A_{11}^{-1}A_{11} + A_{21} & -A_{21}A_{11}^{-1}A_{12} + A_{22} \end{pmatrix}$$

$$= \begin{pmatrix} A_{11} & A_{12} \\ -A_{21} + A_{21} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}$$

$$= \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}$$

(b) If A_{21} is eliminated row by row by means of n steps of Gaussian elimination, we can think of LU factorization to achieve this. In case of LU factorization, when A_{21} will be zero after n steps of Gaussian elimination, by that time A_{11} will be turned into L_{11}^{-1} considering the following formula, $A_{11} = L_{11}U_{11}$. Now, if we consider the whole triangular matrix for A, the bottom left corner $(m-n) \times (m-n)$ matrix of L will become $-A_{21}A_{11}^{-1}$ and bottom right corner $(m-n) \times (m-n)$ matrix of L will become I. So, the L will be,

$L = \begin{pmatrix} L_{11}^{-1} & 0 \\ -A_{21}A_{11}^{-1} & I \end{pmatrix}$

So, finally,

$LA = \begin{pmatrix} L_{11}^{-1} & 0 \\ -A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$

$= \begin{pmatrix} L_{11}^{-1}A_{11} & L_{11}^{-1}A_{12} \\ -A_{21}A_{11}^{-1}A_{11} + A_{21} & -A_{21}A_{11}^{-1}A_{12} + A_{22} \end{pmatrix}$

$= \begin{pmatrix} U_{11} & L_{11}^{-1}A_{12} \\ 0 & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}$(showed)

11. (9 points) Gaussian elimination can be used to compute the inverse A^{-1} of a nonsingular matrix $A \in \mathbb{C}^{m \times m}$, though it is rarely really necessary to do so.
(a) Describe an algorithm for computing A^{-1} by solving m systems of

equations, and show that it is asymptotic operation count is $8m^3/3$ flops.
(b) Describe a variant of your algorithm, taking advantage of sparsity, that reduces the operation count to $2m^3$ flops.
(c) Suppose one wishes to solve n systems of equations $Ax_j = b_j$, or equivalently, a block system $AX = B$ with $B \in \mathbb{C}^{m\times n}$. What is the asymptotic operation count (a function of m and n) for doing this (i) directly from the LU factorization and (ii) with a preliminary computation of A^{-1}?

Answer:
(a) We can solve A^{-1} using the following LU factorization formula,
$A^{-1} = U^{-1}L^{-1}$
Here, to calculate L and U we need $\frac{2m^3}{3}$ flops and for each inverse, we will need $m \times m$ flops for addition and subtraction row wise. So, for m rows, the operations will be, m^3.
So the total number of flops will be required, $m^3 + m^3 + \frac{2m^3}{3} = \frac{8m^3}{3}$ (showed)

(b)Here for each inverse operation, we do not need to calculate m^3 flops, rather we can reduce each inverse operation to $\frac{2m^3}{3}$. So, total flops then will be,
$\frac{2m^3}{3} + \frac{2m^3}{3} + \frac{2m^3}{3} = 2m^3$

(c)
(i)Each column will require $2m^2$ and for n equations it will be $2m^2n$. So total flops will be needed
$2m^2n + \frac{2m^3}{3}$
(ii) For A^{-1} we will need $2m^3$ flops and n equations we will need $2m^2n$ flops. SO total,
$2m^2n + 2m^3$

12. (4 points) Let A be a nonsingular square matrix and let $A = QR$ and $A^*A = U^*U$ be QR and Cholesky factorizations, respectively, with the usual normalizations $r_{jj}, u_{jj} > 0$. Is it true or false that $R = U$? **Answer:** The answer will be true. Here, given A is nonsingular. So, the Q and R we will obtain by factorizing A will be unique and $r_{jj} > 0$.
Now, $A^*A = (QR)^*QR = R^*Q^*QR = R^*R$
Also, as A is a nonsingular square matrix, A^*A is hermatian. So, any vector u not equal 0, $Au! = 0$.

So, we can write, $u^*A^*Au = (Au)^*Au = ||Au||_2^2 > 0$
So, A^*A is also positive definite and any hermatian positive definite metrix has a unique cholesky factorization.
So, $A^*A = U^*U$
So, We can say, $R = U$ (Proved)