CS 513 Assignment 1

Ruochen Lin

February 13, 2018

1

1.1

1.1.1

From linear algebra, we know that to apply a column operation on B, we can multiply B by a matrix on its right; similarly, to apply a row operation, we can multiply B on its left. In addition, if the desired outcome of a row operation on B_0 is B_1, and the matrix corresponding to the operation is A, i.e. $AB_0 = B_1$, we can know about the structure of A by applying it onto identity matrix I. For example, if the desired operation is halving row three and the matrix is A, then

$$A = AI = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \frac{1}{2} & \\ & & & 1 \end{bmatrix}.$$

Apparently, this conclusion can be generalised to multiple consecutive row (column) operations.

With this in mind, we can write out the matrices corresponding to operations 1 through 7. Note that matrices corresponding to row operations are denoted with A_i, and column operations C_i.

$$C_1 = \begin{bmatrix} 2 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \frac{1}{2} & \\ & & & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$C_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

$$C_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C_7 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And the resulting matrix is $A_5A_3A_2BC_1C_4C_6C_7$.

1.1.2

$$A = A_5A_3A_2 = \begin{bmatrix} 1 & -1 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & \frac{1}{2} & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

$$C = C_1C_4C_6C_7 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

See Appendix A for Matlab code that verifies the results above.

1.2

Since A is Hermitian, $A = A'$, where A' refers to the adjoint of A. Suppose $x \in \mathbb{R}^m$ is an eigenvector of A with eigenvalue λ: i.e. $Ax = \lambda x$. On one hand,

$$(x, Ax) = (x, \lambda x) = \lambda(x, x);$$

on the other,

$$(x, Ax) = (A'x, x) = (Ax, x) = (\lambda x, x) = \lambda^*(x, x).$$

Since $\lambda(x, x) = \lambda^*(x, x)$ and $x \neq 0$, we know $\lambda = \lambda^*$, and hence all eigenvalues of A are real.

If $Ax = \lambda_x x$ and $Ay = \lambda_y y$, $\lambda_x \neq \lambda_y$, then $(x, Ay) = \lambda_y(x, y)$ and $(x, Ay) = (A'x, y) = (Ax, y) = \lambda_x(x, y)$. Since $\lambda_x \neq \lambda_y$, $(x, y) = 0$. This is to say that eigenvectors of the same matrix that correspond to different eigenvalues are orthogonal.

2

2.1

We know that $\|Qx\|_2 = \|x\|_2 \Leftrightarrow (Qx, Qx) = (x, x)$. In addition, $(Qx, Qx) = (x, Q'Qx)$. Suppose x is an eigenvector of $Q'Q$ with eigenvalue λ, then $(x, Q'Qx) = (x, \lambda x) = \lambda(x, x)$, and hence $\lambda(x, x) = (x, x)$. Since $x \neq 0$, we know that $\lambda = 1$, $\forall \lambda \in \sigma(Q'Q)$.

2.2

Write Q as $\begin{bmatrix} q_1 & q_2 & \dots & q_m \end{bmatrix}$, then

$$Q'Q = \begin{bmatrix} q_1' \\ q_2' \\ \vdots \\ q_m' \end{bmatrix} \begin{bmatrix} q_1 & q_2 & \dots & q_m \end{bmatrix};$$

$(Q'Q)_{ij} = (q_i, q_j)$, $(Q'Q)_{ji} = (q_j, q_i) = (q_i, q_j) = (Q'Q)_{ij}$. Hence $Q'Q$ is symmetric.

Since $Q'Q$ is real and symmetric, it can be written as $Q'Q = P\Lambda P'$, with P being orthogonal and Λ diagonal with eigenvalues of Q as its entries. In addition, we've proved 1 is the only eigenvalue of $Q'Q$; so $\Lambda = I$ in this particular case. Insert this into the decomposed form of $Q'Q$, we have $Q'Q = PIP' = PP' = I$. In other words, $Q^{-1} = Q'$; hence we proved Q is orthogonal.

3

The matrix multiplication at the end of the for loop in sloppy_qr.m is the most tedious computation in the loop, and it is $O(n^3)$. In addition, the for loop runs from 1 through n; thus we would expect $O(n^4)$ complexity from the "sloppy" implementation of QR-factorization.

With this in mind, the sloppy_qr.m code was tweaked to count the operations needed for different ns. For each $n = 10, 20, \ldots, 100$, we generated a square matrix that has random entries. The number of operations needed to QR-factorize the matrix is counted for each n by summing up the number of operations suggested in comments. Fitting the number of operations versus n, we have the following expression:

$$\text{Complexity}(n) = 2n^4 - 0.00649n^3 + 3.407n^2 - 9.458n + 58.33.$$

The modified version of code and raw data can be found in Appendix B and C. Note that the statements at the bottom of the original code was not taken into our consideration of computational complexity, since they are verification of our implementation QR-factorization, rather than part of the decomposition. In addition, the $O(n^4)$ complexity can also be verified from the fact that, if we do choose to use "`poly5`" curve fit, the coefficient of the n^5 term will be small and can be neglected.

A Matlab Code to Verify Results in Problem 1

```
C1=[2 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
A2=[1 0 0 0; 0 1 0 0; 0 0 1/2 0; 0 0 0 1];
A3=[1 0 1 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
C4=[0 0 0 1; 0 1 0 0; 0 0 1 0; 1 0 0 0];
A5=[1 -1 0 0; 0 1 0 0; 0 -1 1 0; 0 -1 0 1];
C6=[1 0 0 0; 0 1 0 0; 0 0 1 1; 0 0 0 0];
C7=[0 0 0; 1 0 0; 0 1 0; 0 0 1];
A=A5*A3*A2;
C=C1*C4*C6*C7;
```

And the results was:

```
A =

1.0000 -1.0000 0.5000 0
0 1.0000 0 0
0 -1.0000 0.5000 0
0 -1.0000 0 1.0000
```

and

```
C =

0 0 0
1 0 0
0 1 1
0 0 0,
```

hence verifying our results in problem 1.

B Matlab Code for Problem 3

```
count=zeros(1,10)
for j=1:10,
dim=j*10;
A=zeros(dim,dim);
for a=1:dim,
for b=1:dim,
A(a,b)=100*rand();
end
end
[m,n]=size(A);
R=A;
Q=eye(m);
for i=1:n,
x=R(:,i);
a=norm(x(i:m),2);
count(1,j)=count(1,j)+n;
y=[x(1:i-1)’ a zeros(1,m-i)]’;
w=x-y;
count(1,j) = count(1,j)+n;
if norm(w) =0,
w=w/norm(w);
count(1,j) = count(1,j)+n;
end
H=eye(m)-2*w*w’;
Q=Q*H; R=H*R;
count(1,j) = count(1,j)+2*n^3;
end
% norm(A-Q*R), norm(eye(m)-Q’*Q)
end
complexity=fit(linspace(10,100,10)’,count’, ’poly4’);
```

I apologize for the (lack of) indentation; I’ve tried (and obviously failed) to get the correct indentation with fixed-width fonts in LaTex.

C Raw Output of Problem 3

The results of the code in Problem 3 yields the following output:

```
count =
20290 321200 1622700 5124800 12507450 25930800 48034700 81939120
131244210 200030000

Linear model Poly4:
complexity(x) = p1*x^4 + p2*x^3 + p3*x^2 + p4*x + p5
Coefficients (with 95% confidence bounds):
p1 = 2 (2, 2)
p2 = -0.00649 (-0.02165, 0.008673)
p3 = 3.407 (2.272, 4.542)
p4 = -9.458 (-42.22, 23.31)
p5 = 58.33 (-232.4, 349)
```