Damian Franco

dfranco24@unm.edu

101789677

CS-575

Homework 7

1) For the first problem we were asked to show that the 2-norm of *A* is equal to the largest eigenvalue of *A* (λ_{max}) and that the 2-norm of A^{-1} is equal to one over the smallest eigenvalue of *A* (λ_{min}). knowing that the matrix *A* was SPD, I was able to use SPD properties alongside the definition and properties of eigenvalues. Knowing the *A* is SPD and a *mxm* matrix, I first know that there is an eigenvector *x* that is the corresponding eigenvalue λ_{max}. I first used that property and the fact that the eigenvector *x* is a unit vector to show that the 2-norm is in fact equal to the largest eigenvalue of *A* (λ_{max}). I used the same approach to show that the 2-norm of A^{-1} is equal to one over the smallest eigenvalue (λ_{min}) with a unit eigenvector *x*, but used more SPD, inverse properties, norm properties and FOIL rules to expand on how the 2-norm of the inverse matrix of *A* is equal to 1 over the smallest eigenvalue (λ_{min}) of *A*. My work is shown below and on the next page.

1) Let $A \in \mathbb{R}^{m \times m}$ be a symmetric positive definite matrix.
Let λ_{max} and λ_{min} be the largest and smallest eigenvalues of A respectively.
Show that:

$$||A||_2 = \lambda_{max} \text{ and } ||A^{-1}||_2 = 1/\lambda_{min}$$

$$\begin{aligned}
||A||_2 &= ||Ax||_2 \\
&= ||\lambda_{max} x||_2 \\
&= |\lambda_{max}| \cdot ||x||_2 \\
&= |\lambda_{max}| \cdot 1 \\
&= |\lambda_{max}| \\
&= \lambda_{max} \quad \checkmark \text{DONE}
\end{aligned}$$

$$\|A^{-1}\|_2 = \|A^{-1}x\|_2$$
$$= \|A^{-1}x\|_2^2$$
$$= x^T(A^{-1})^T A^{-1} x$$
$$= x^T A^{-1} A A^{-1} x$$
$$= x^T(A^{-1})^T \lambda_{min}^{-1} A^{-1} x$$
$$= \lambda_{min}^{-1} x^T (A^{-1})^T A^{-1} x$$
$$= \lambda_{min}^{-1} x^T x (A^{-1})^T A^{-1}$$
$$= \lambda_{min}^{-1} I \; \underline{I}$$
$$= \lambda_{min}^{-1}$$
$$= 1/\lambda_{min} \checkmark$$ DONE

Therefore $\|A\|_2 = \lambda_{max}$

and

$\|A^{-1}\|_2 = 1/\lambda_{min}$

2) Next, we were asked to show that $B(Sx) = \lambda(Sx)$ where A and B are both complex mxm matrices and S is an invertible matrix. We are given two properties. The first is showing the invertible matrix S and how it relates to matrices A and B. The next shows an eigenvalue and eigenvector property where x is the eigenvector of m length associated with λ. Initially, I was thinking of starting with left side $B(Sx)$, but we were given the rule where A is equal to the inverse of S multiplied to B and S, so I decided to start with $Ax = \lambda x$ which has A in the equation so I can first apply our property that is given to us. Next, I multiplied both sides by S, and used identity properties to cancel terms and combined terms to find that $B(Sx)$ is in fact equal to $\lambda(Sx)$. You can see my work on the next page.

2) Let $A \in \mathbb{C}^{m \times m}$, $B \in \mathbb{C}^{m \times m}$ be similar and $S \in \mathbb{C}^{m \times m}$ be an invertible matrix such that $A = S^{-1}BS$

Let $x \in \mathbb{C}^m$, $x \neq 0$ and $\lambda \in \mathbb{C}$ such that

$$Ax = \lambda x$$

Show that

$$B(Sx) = \lambda(Sx)$$

$$Ax = \lambda x$$

$$Ax = \lambda x$$

$$(S^{-1}BS)x = \lambda x$$

$$S^{-1}BSx = \lambda x$$

$$S \cdot (S^{-1}BSx) = S \cdot (\lambda x)$$

$$SS^{-1}BSx = S\lambda x$$

$$I\,BSx = S\lambda x$$

$$BSx = \lambda Sx$$

$$B(Sx) = \lambda(Sx) \quad \checkmark \text{DONE}$$

Therefore $B(Sx)$ is equal to $\lambda(Sx)$ where x is an eigen vector of A and λ is an eigen value of A.

3) For our last problem in this homework, we were asked to write a program that finds eigenvalues and the associated eigenvectors with it. My program implements two versions of the power iteration method to predict the largest eigenvalue and its associated eigenvector. The first implementation is a naive version with no normalization techniques while the second implementation utilizes normalization techniques to properly compute the eigenvectors associated with the largest eigenvalues predictions. The reason for two versions of the power iteration method was the first implementation I did had very interesting errors when computed. Specifically, the eigenvalue errors that were computed were extremely large and never changing. There seemed to be an issue with the way I implemented it so I decided to look more into it and found that I was not doing any normalizations which led to a new implementation of implementing a version that normalized which resulted in more efficient error calculations. I noticed that the errors for both the eigenvalue and eigenvector seem to be converging with a smaller error after 18 iterations. That is when the error factor to both the eigenvalue and eigenvector is smaller so convergence is occurring around those iterations. The error factors generated seem to behave as predicted when it comes to the eigenvalue errors. It seems to converge and get lesser as iterations continue. Eigenvector error factor is interesting because in my normalized version of the power iteration, I notice that each time the error factor is around 3 or 4 and does not change from those and fluctuates from 3 to 4. I am not sure why this is occurring, but it is very interesting. The eigenvalue is converging to around 19.549. My two Python implementations and tables are shown in the next pages, with the non-normalized version shown first and the normalized version shown next.

Naive Power Method:

```python
# Power iteration method, naive approach, not normailzed
def powerIter(A, x, tol):
  lambdaList = []
  xList = []
  for i in range(tol):
    x = np.dot(A, x)
    # Compute eigenvalue
    eigenVal = abs(x).max()
    # Compute eigenvector
    x = x / x.max()
    lambdaList.append(eigenVal)
    xList.append(x)
  return eigenVal, x, lambdaList, xList
```

Naive Power MethodResults:

Iteration	Estimated EigenVal	EigenVal Error	EigenVal Error Factor	EigenVec Error	EigenVec Error Factor
1	14.0	24.939156772997116	0.0	0.1286530925797249	0.0
2	18.142857142857142	32.173953061467245	0.7751349896405861	0.029932263937094365	4.298141057759687
3	19.889763779527563	30.435747150901015	1.0571106699614823	0.008128716099952565	3.6822867927776484
4	19.464568487727632	30.879333173563474	0.9856348574572775	0.0016578661772166062	4.903119571206813
5	19.568210708293076	30.783430251490394	1.0031154072593464	0.0004766160822021834	3.4784100644621754
6	19.54449964661373	30.808979481837987	0.9991707213034221	9.301002172728282e-05	5.124351906934096
7	19.550416998286657	30.803646161896914	1.0001731392417976	2.8165325805999325e-05	3.302288152742452
8	19.549084736099694	30.80511787711462	0.9999522249769154	5.255613494465686e-06	5.359093821426981
9	19.549422319904124	30.804822741622367	1.000009580820988	1.6883327414555746e-06	3.1129014828763113
10	19.549347589288363	30.80490790009833	0.999997235554924	3.013901238579586e-07	5.601818400165178
11	19.549366887769906	30.804891691439515	1.0000005261715892	1.0305852501638279e-07	2.924456019626206
12	19.549362707883823	30.804896647986595	0.9999998390987271	1.7764828482641817e-08	5.801267663072697
13	19.54936381418296	30.80489576716866	1.0000000285934398	6.4283947254381595e-09	2.763493724544267
14	19.549363581289192	30.8048960577514	0.999999990566995	1.0923975086780996e-09	5.8846662266898635
15	19.549363644933315	30.804896010595648	1.0000000015307875	4.1063815303015513e-10	2.6602435760465726
16	19.54936363202401	30.804896027781275	0.9999999994421138	7.05551966737965e-11	5.820097914668021
17	19.549363635701724	30.804896025311223	1.0000000000801839	2.6735304931252855e-11	2.6390271910203413
18	19.54936363499118	30.804896026338277	0.9999999999666593	4.503518871251187e-12	5.936536671783755
19	19.549363635204884	30.804896026213136	1.0000000000040623	1.6263547649039027e-12	2.7690876360037526
20	19.549363635166152	30.80489602627525	0.9999999999979836	0.0	0.0

Normalized Power Method:

```python
# Normalized version of power iteration method
def powerIter_Normalized(A, x, tol):
    x_curr = x
    x_curr = x_curr / np.linalg.norm(x_curr) # Intial normalization
    eigenVal = 0
    eigenVec = np.zeros(tol)
    valList = []
    vecList = []
    for i in range(tol):
        y = np.matmul(A, x_curr)
        # Compute eigenvalue
        currEigenVal = np.dot(x_curr, y)
        eigenVal = currEigenVal
        valList.append(eigenVal)
        # Compute eigenvector
        currEigenVec = y / np.linalg.norm(y)
        eigenVec = currEigenVec
        vecList.append(eigenVec)
        # Update current eigenvector
        x_curr = eigenVec

    return eigenVal, eigenVec, valList, vecList
```

Normalized Power Method Results:

Iteration	Estimated EigenVal	EigenVal Error	EigenVal Error Factor	EigenVec Error	EigenVec Error Factor
1	14.0	0.06388877483095712	0.0	0.056474546861129425	0.0
2	18.142857142857142	0.00038400148454087457	166.37637457924086	0.012768379455168599	4.423000354854642
3	19.889763779527563	0.00216163124039781	0.1776443073936172	0.0034357045390233805	3.7163787834905286
4	19.464568487727632	0.00028074221252083476	7.69970151972564	0.0007289289507268106	4.713359972323311
5	19.568210708293076	0.0001661138285129482	1.690059250539467	0.00020231481158017772	3.602944070350157
6	19.54449964661373	2.4485663875140062e-05	6.784125983269792	4.20106060087738e-05	4.815803217357179
7	19.550416998286657	1.1602080263628523e-05	2.110454618375673	1.210384997107185e-05	3.470846557845559
8	19.549084736099694	1.9716423231841418e-06	5.88447515414029	2.4678981224583485e-06	4.904517678799008
9	19.549422319904124	8.152802948302451e-07	2.4183613116697127	7.391482731971982e-07	3.338840408546737
10	19.549347589288363	1.5459076863066912e-07	5.273796760646304	1.49269597325479e-07	4.951767047280914
11	19.549366887769906	5.7822482801839215e-08	2.673540829446222	4.627393432864414e-08	3.225781414334577
12	19.549362707883823	1.1925536114176793e-08	4.848627537432152	9.38520693200393e-09	4.930518278808342
13	19.54936381418296	4.142375331639414e-09	2.8789124981238876	2.9781336040662133e-09	3.15137202682572
14	19.549363581289192	9.185505689401907e-10	4.509686751834275	6.160954825090357e-10	4.833883202548131
15	19.549363644933315	3.0819080620858585e-10	2.9804606446258126	1.9698968857660358e-10	3.1275519391942894
16	19.54936363202401	7.962341896927683e-11	3.870605033018026	4.1929989472355166e-11	4.698061961271913
17	19.549363635701724	3.256417357988539e-11	2.4451232816932142	1.3238290062761579e-11	3.1673266920099756
18	19.54936363499118	1.6228796084760688e-11	2.0065674255691768	2.7605936331778352e-12	4.795450479802211
19	19.549363635204884	1.2654766123887384e-11	1.282425603593487	7.800910609569593e-13	3.5388094689757614
20	19.549363635166152	1.1468159755168017e-11	1.1034696406443618	0.0	0.0