CENG 371 - Scientific Computing

Fall 2022

Homework 3

Anıl Utku Ilgın
2260073

December 26, 2022

Question 1

c) Find the largest and smallest –in magnitude– eigenvalues and corresponding eigenvectors of

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

For $\quad$ v = [[1.] [1.] [1.] [1.] [1.]]^T

Largest eigenvalue in magnitude is
3.73205081

And Corresponding largest eigenvector is :

[[0.28867519] [-0.5] [0.57735021][-0.5] [0.28867519]] ^{T}

Smallest eigenvalue in magnitude is
0.26794919

And corresponding smallest eigenvector is :

[[0.28867514] [0.5] [0.57735027] [0.5] [0.28867514]]^T

I found the *largest one* by using **A** as input to power_method function and **A**$^{\mathbf{-1}}$ as input to find the *smallest one.*

d) Find the largest eigenvalue and eigenvector of

$$B = \begin{bmatrix} 0.2 & 0.3 & -0.5 \\ 0.6 & -0.8 & 0.2 \\ -1.0 & 0.1 & 0.9 \end{bmatrix}$$

Starting with $\mathbf{x} = [[1], [1], [1]]^T$

- At the pen and paper part, after the first iteration I ended up with

  $\mathbf{x_1} = [\ [0], [0], [0]\ ]$

  Therefore I couldn't continue with the power method iteration.

However, in the python script I ended up with the correct eigenvalue and eigenvector, respectively

$e_1 = 1.34268604$ and $v_1 = [[-0.40703133]\ [-0.02876139]\ [\ 0.91296127]]^T$

I think this is doable on computers because of floating point errors. I debugged the code and at the end of the first iteration, I saw $x_1 = [\ [\ 0\ ], [-5.55111512e-17], [\ 0\ ]]$. With this slight difference from 0, the computer managed to approximate eigenvalue and eigenvector correctly.

Question 2

a) İntuitively, by subtracting $Lambda * (V_i\ V_i^T)/(\ V_i^T\ V_i)$ from A, we are removing the ith eigenvalue from the A's eigenspace. Therefore, in the next iteration when we apply the power method, we find the next eigenvector. However I couldn't do the math to prove this.

d) In my implementations, I tried my functions with the can_299.mtx file using scipy.io to convert them into numpy matrices. I tried with the comment-outed parts in my code, and I see subspace_iterations were slower than the power_k. Since subspace iterations were doing QR factorization iteratively, I think that subspace iterations should be much slower than the power_k method.. When I compared the results with the first 5 elements, I saw that subspace_iterations were in order of largest to smallest, yet power_k was inordered. Therefore I think my implementation was not correct. However, subspace_iterations was still slower due to multiple QR factorization operations.

Therefore, theoretically I think Subspace_iterations should be slower since there is at least one QR factorization in each loop and overall loop count is significantly higher than the power_k.

Time Results for

Algorithm / k	k = 5	k = 100	k = 200
Power_k	0.018675	0.29645	0.29975
Subspace Iteration	0.292903	0.28578	0.47078