Jack Sutton Dienhart

Avery, Scientific Computing

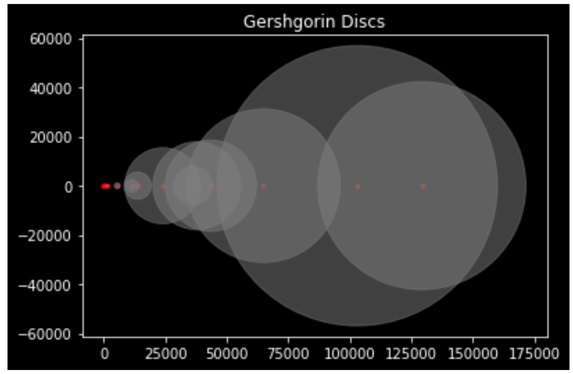
27/09/22

Project 2 Writeup

[A]

1. and 2.

My function implements Gersghorin's disc theory, finding the scope of each eigenset. I visualized my results on the figure to the right.




```python
def gershgorin(A):
    n = len(A)
    radius = np.zeros((n,1))
    points = np.zeros((n,1))
    for i in range(n):
        radius[i]=sum(abs(A[i,:]))-abs(A[i,i])
        points[i]=A[i,i]
    return points,radius
```

[B]

1. The rayleigh quotient was fairly simple to create (note: I chose to ensure x^H instead of just x^T for Hermitian cases)

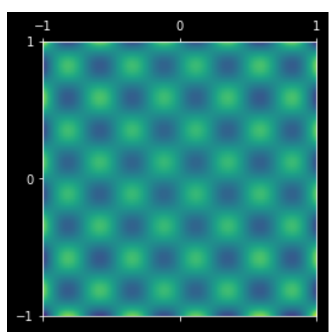
```python
def rayleigh_qt(A,x):
    xt = x.conjugate().T
    ral = xt.dot(A).dot(x)/xt.dot(x)
    return ral
```

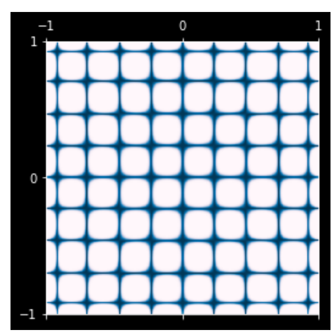
2. The power iteration was fairly straightforward as well. I had a few different methods of error criterion I was using. The power iteration operates on convergence to 0 for all (lambda(i!=1)/lambda(1))^k where k -> infinity, so testing that ensures the method is working, but for accuracy I also tested the results (both norm and Rayleigh values) by their residual errors.

```python
def power_iterate(A,x0):
    x = np.random.rand(len(A))
    xn = np.random.rand()
    k = 0
    while(ercrit(A,x,xn)==False):
        x = A.dot(x)
        xn = np.linalg.norm(x)
        x = x/xn
        k+=1
    return x, xn, k
```

3. I found all the largest values, with comparative errors and tables (see part C)

4. The largest eigenvalue was about 1.51e05. Using the library functions included I got the following plots:





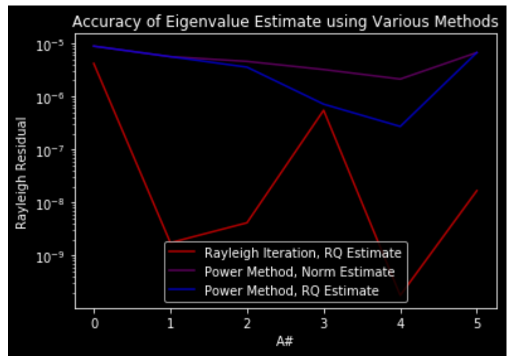
[C]

1. Here I chose to use my LU factorization to implement the Rayleigh Quotient Iteration, which takes a matrix, our guess vector, and guess shift (or eigenvalue) and returns the best approximation eigenvalue, vectors, and how many steps it took (with a pre-determined error of 10**(-i) minimum regression test added to the output in the case of C2).

```python
def r(A,b,u):
    n = len(A)
    Id = np.eye(n)
    k = 0
    while(ercrit(A,b,u,3)==False):
        u = b.T.dot(A).dot(b)/np.dot(b.T,b)
        LHS= A-u*Id
        L, U, Lin = lu_factorize(LHS)
        newB = forward_substitute(L,b)
        newB = back_substitute(U,newB)
        bn= np.linalg.norm(newB)
        b = newB/bn
        k+=1
    return u, b, k
```

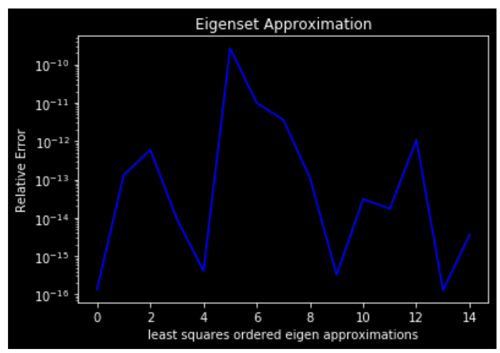
2. Below I show my test results using both Power and Rayleigh iteration (with other desired values) and compare their errors.

```
Eigenvalues, Iteration Count, and Rayleigh Residuals using Power Method

[[3.9999999999790131, 15, 8.871352254129581e-06],
 [3.9999999992431152, 16, 5.615769863390635e-06],
 [12.298932521781474, 12, 3.5657723944570656e-06],
 [16.116891239185705, 5, 7.149745617806083e-07],
 [68.64221815576249, 5, 2.7146544452725924e-07],
 [1.999999999458957, 16, 6.71468263202117e-06]]
Eigenvalues, Iteration Count, and Rayleigh Residuals using Rayleigh Iteration

[[array([[3.99998329]]), 2, array([4.17868891e-06])],
 [array([[3.99999999]]), 2, array([1.7453052e-09])],
 [array([[12.29895844]]), 4, array([4.14786169e-09])],
 [array([[16.11685282]]), 5, array([5.49146214e-07])],
 [array([[68.64208075]]), 5, array([1.77366053e-10])],
 [array([[1.99999997]]), 3, array([1.67313739e-08])]]
```

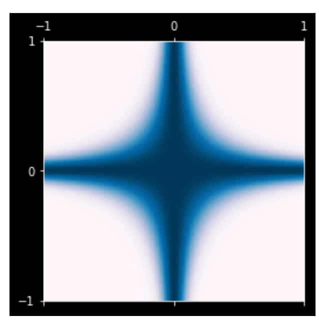


[D]

1. The principle of power iteration is that we are using the fact that ordering our eigenvalues lambda1 -> lambdaN from highest to lowest (in a linear combination), then factoring out lambda1 from both sides helps us built the correct highest eigenvector/value, since each other ratio of values (lambdak/lambda1)^k will approach 0 as k -> infinity (iterated interaction). By definition, this dominance therefore eliminates smaller eigenvalues, so one cannot artificially decide to recover the "middle" ones using this simple method.



2. I used the discs to center my approximate eigen-guesses to successfully replicate (to relative errors that are quite small) all eigenvalues of Kmat, and tested by showing my lowest eigenvector with the show_nodes library function:



3. I created a T matrix built up out of my eigenvalues, its inverse, and then used an identity iteratively placing my eigenvalues at each diagonal. I tested to make sure this RHS error was nonimpactful compared to Kmat. This composition used my index from (D2), which stored all the vectors, values, residuals, and k-counts of each set.

4. I adjusted my index slightly (needed to be squeezed) and simply used the library function to see all the nodes