SE 284 - Numerical Linear Algebra
Homework 3

Aakriti Gupta (09237)

Take a 30x30 random matrix R and form A = R^TR

```
function A = Random

R = randn(30,30);
A = R'*R;
```

1. Write a computer program to find the largest Eigen value and corresponding Eigen vector of A using traditional power method. Does your result match up with the values obtained using the function eig in MATLAB. Defend your answer. Is your largest Eigen value real? If yes why?

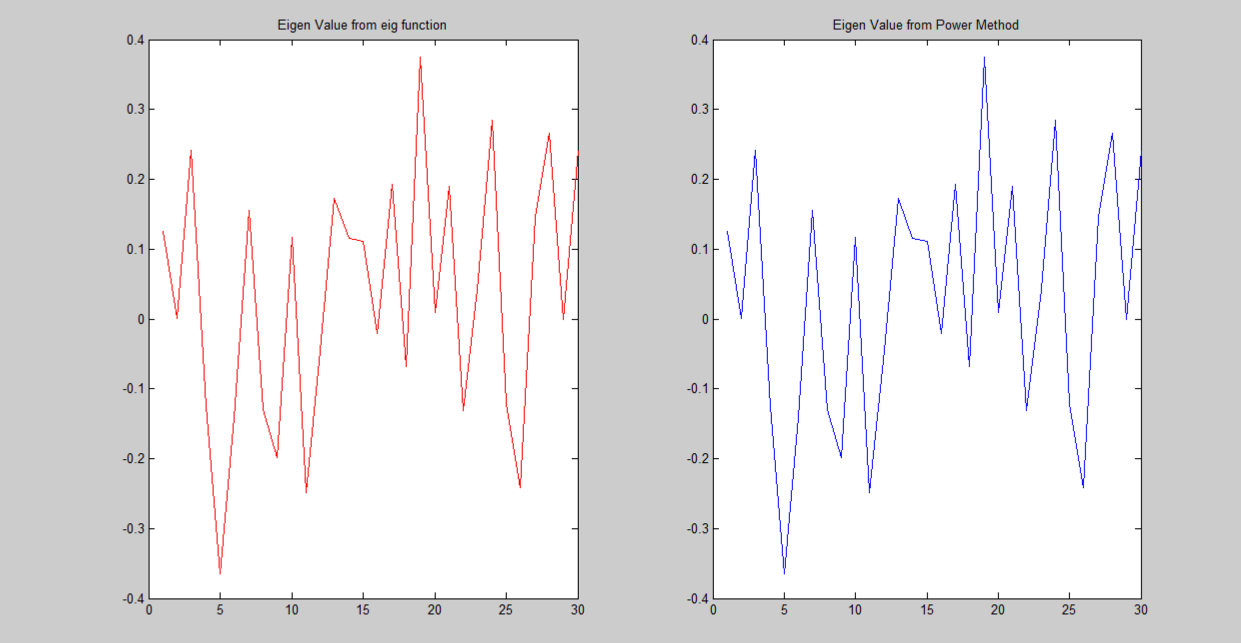
Code for largest Eigen value using Power Method:

```
function [val Z] = PowerMethod(A)
[n ~] = size(A);
x = ones(n,1);

for i = 1:1000
      Z = A*x;
      s = norm(Z);
      Z = Z/s;
      if norm(Z-x)<1e-6 || norm(Z+x)<1e-6
            break;
      end
x = Z;
end

val = (Z'*A*Z)/(Z'*Z);
```

Code for largest Eigen value using MATLAB eig function:

```
function [val Z] = eigMax(A)

[n ~] = size(A);
[Z val] = eig(A);

val = val(n,n);
Z = Z(:,n);
[v1 Z1] = PowerMethod(A); % For comparison
```

The result matches with the result computed by the MATLAB eig function.



```
>> norm(Z-Z1,2)

ans =

6.1232e-006
```

Largest Eigen value is real. Since A is a real symmetric matrix, its Eigen values are real with real Eigen vectors. Proof:

Given, $A = A^H$ (For real matrices $A^H = A^T$)

Let x be an Eigen Vector of A and λ be the corresponding Eigen value,

We have $Ax = \lambda x$
$x^H Ax = x^H \lambda x$
$x^H Ax = \lambda \ ||x||_2$ (L2 Norm)

Take Hermitian of L.H.S.

$(x^H Ax)^H = x^H A^H x = \lambda^* \ ||x||_2$

But $A = A^H$
So,
$\lambda \ ||x||_2 = \lambda^* \ ||x||_2$

Since x is non-zero vector, we have

$\lambda = \lambda^*$
Hence λ is real.

2. Do same as above using pure QR algorithm. Compare and contrast both algorithms in terms of number of iterations, operations count, and run time.

Code for QR Decomposition:

```
    function [Q R] = QR(A)

    [m n] = size(A);
    I = eye(n);
    A = [A I];

    for j = 1:n
        for i= j+1 : n
            d = A(i,j)*A(i,j) + A(j,j)*A(j,j);
            d = sqrt(d);
            c = A(j,j) / d;
            s = A(i,j) / d;
            for k = j:2*n
                t = A(j,k);
                A(j,k) = c*t + s*A(i,k);
                A(i,k) = -1*s*t + c*A(i,k);
            end
        end
    end
    R = A(:,1:n);
    Q = A(:,n+1:2*n);
    Q = Q';
```

Code for computing all Eigen Values using Basic QR iterations:

```
    function D = BasicQR(A)

    for i = 1:1000
        [Q R] = QR(A);
        A1 = R*Q;
        if norm(A-A1)<1e-6 || norm(A+A1)<1e-6
            break;
        end
        A = A1;
    end
    D = diag(A1);
```

For comparison, we have to modify this code to get the maximum eigen value only.
Code for computing max Eigen Value using Basic QR iterations:

```
    function D = BasicQR(A)

    for i = 1:1000
    [Q R] = QR(A);
```

```
        A1 = R*Q;
        if norm(A(1,1)-A1(1,1))<1e-6 || norm(A(1,1)+A1(1,1))<1e-6
              break;
        end
        A = A1;
        end
        D = diag(A1);
```

Table of Comparison:

Comparison	Power Method	QR method (finding largest Eigen value)
No of iterations	74	59
Time Taken	0.000776 s	0.554042 s
No. of operations per iteration	O(n^2)	O(n^3)

For number of operations per iteration note that in QR method, QR decomposition is being called in each iteration which is O(n^3) being the dominant step. Whereas in Power Method the max cost operation is a matrix multiplication with a vector and norm calculation which is O(n^2).

3. If one is interested in writing shift power and shift QR algorithms, how do you choose these shifts? Defend your choice. What is the difficulty in implementing the same?

Power Method: Suppose, we choose shift k, and change our matrix A to A-kI, where k is scalar.

Let μ be a eigen value of this new matrix, A - kI.

Therefore, λ = μ+k is one of the eigenvalues of A.

This way shifting in power method can be done. If we chose inverse power method, the eigen value λ will be the eigen value of A closest to the shift value k since μ is smallest Eigen value. This way we can find eigen values using shift power method.

Difficulty: Since we do not know the eigen values, choosing shift values is difficult.

For QR method: shift is selected to reduce the no. of iterations done in basic QR method. First reduce A to an appropriate compact form – Unreduced Hessenberg's Matrix.

To accelerate the convergence, we need to include origin shifts. That is, we first we first choose a shift k_i, then we factor $A_i - k_iI = Q_iR_i$ and compute $A_{i+1} = k_iI + R_iQ_i$.

Note that,

$$A_{i+1} = k_iI + R_iQ_i = k_iI + Q_i^T(A_i - k_iI)Q_i = Q_i^TA_iQ_i$$

So including origin shift does not change the fact that we are generating sequence of matrices orthogonally similar to A.

A very simple shift strategy would be to pick $k_i = a_{nn}$. A more complicated would be to take the Eigen value of the 2x2 lower right hand corner that is nearest a_{nn}.

Difficulty: A given strategy will converge to an upper triangular matrix only if none of the Eigen values of A are of equal modulus.

Reference: An introduction of Numerical Linear Algebra, Charles Cullen.
Chapter 4 (Page 150).

4. Can you find the singular value decomposition of A by posing it as an Eigen value problem? Write a computer program to do the same. Does your result match up with the decomposition obtained using the function svd in MATLAB. Defend your answer.

Yes, we can find SVD of A by posing it as an Eigen value problem. This is because A is a symmetric positive definite matrix. (Since $A = R^TR$).
Hence, we can diagonalize A (using eig function) as $A= QDQ^T$ (from the spectral theorem).
Also since A is ***psd,*** D is a diagonal matrix with positive or zero diagonals.
Hence the form QDQ^T is a valid SVD of A.

```
function [q d q] = EigSVD(A)

[n ~] = size(A);
[q1 d1] = eig(A);
[u s v] = svd(A); %For comparison

for i = 1:n
      q(:,i) = q1(:,n+1-i);
      d(n+1-i,n+1-i) = d1(i,i);
end
```

For symmetric positive definite matrix, eigen values are equal to the singular values.

We have:

$$A = U\ S\ V^T$$

For symmetric A:

$$A^2 = A^TA = V\ S^2\ V^T = AA^T = U\ S^2\ U^T$$

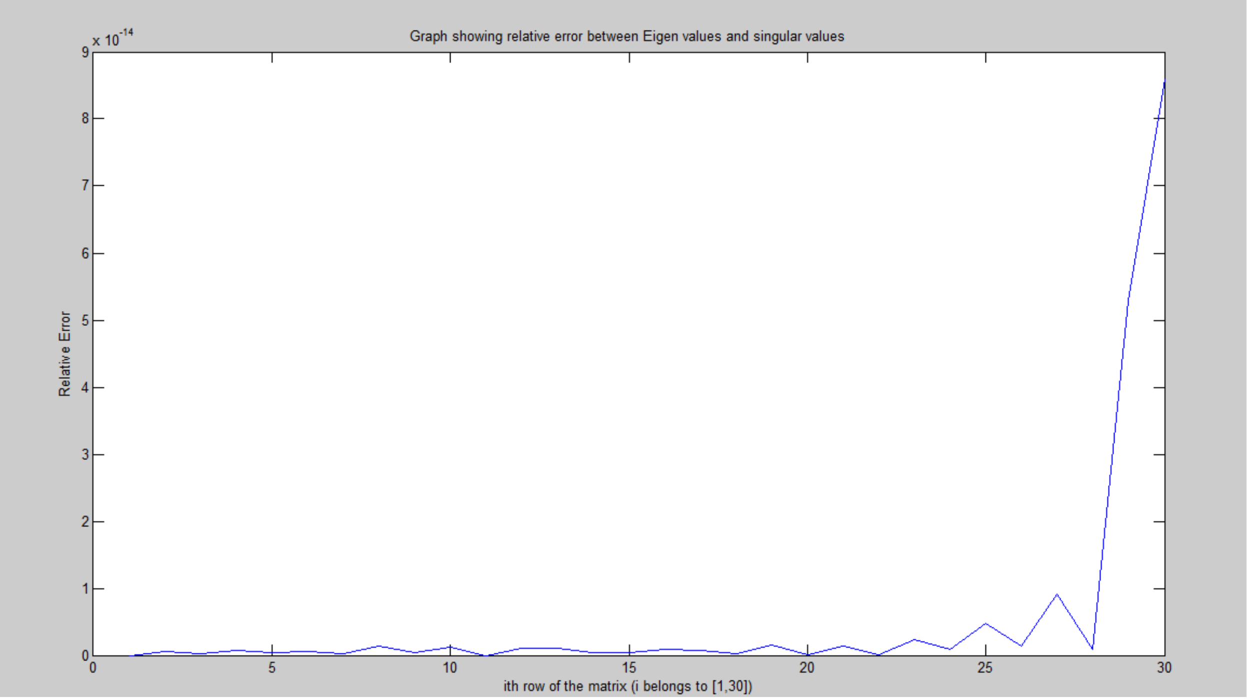
Comparison with the svd function in MATLAB:

Since eig produces the eigen values in increasing order and svd computes singular values in decreasing order, the diagonal matrix of eigen values is changed to match the values as in diagonal matrix from svd and corresponding eigen vector matrix is also changed (column interchanges).

We find that the Eigen values of A are equivalent to the singular values of A

```
      >> norm(diag(d)-diag(s),2)

      ans =

      1.3308e-013
```

Relative error is in the order of 1e-14.



For the eigen vectors and the s & v decomposition in MATLAB we find the following results:

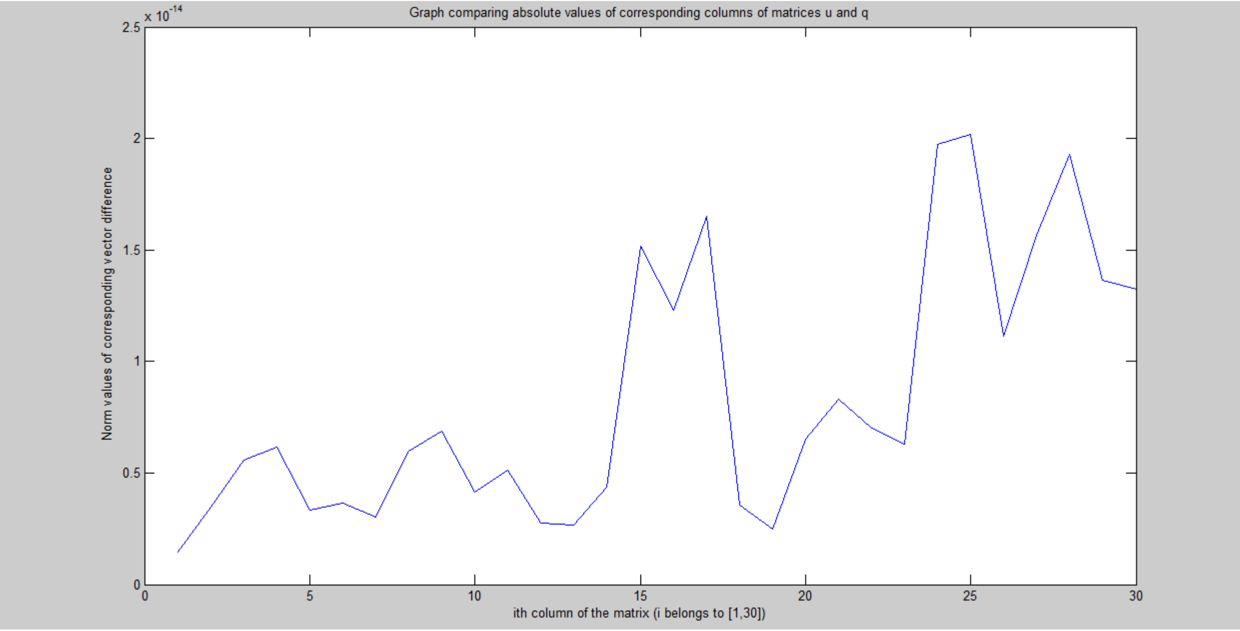
In the SVD, matrix u and v are coming out to be same.

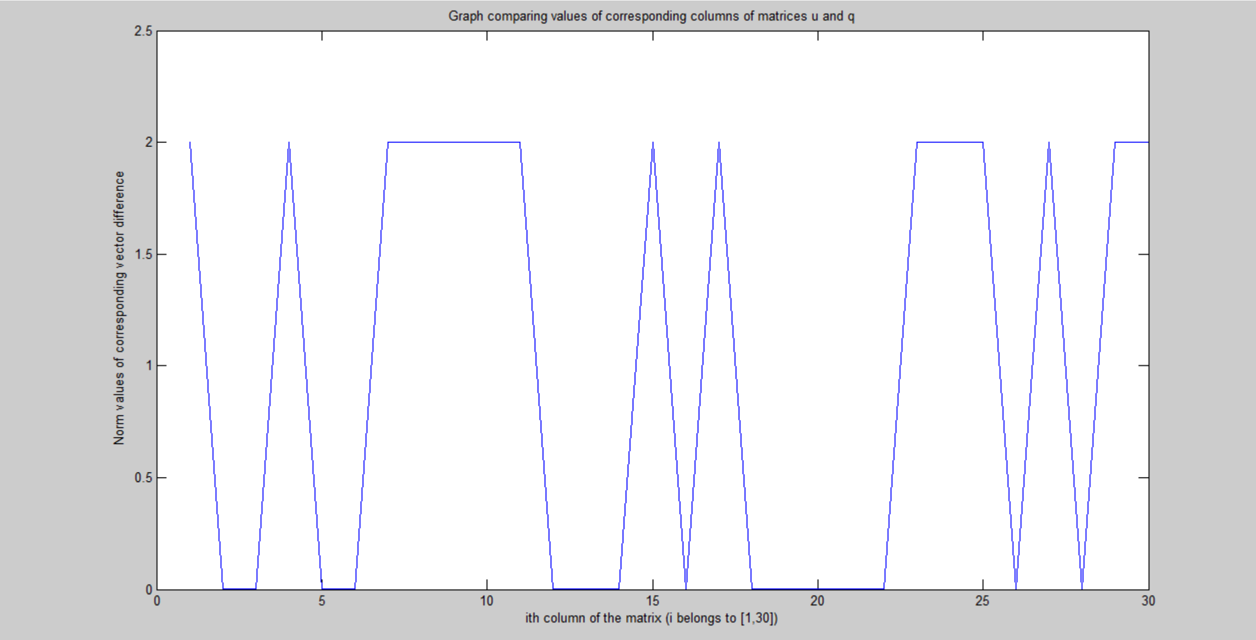
```
        >> norm(u-v,2)

            ans =

            4.3762e-014
```

For matrix q & u we observe that the corresponding columns are similar in magnitude but sometimes opposite in sign.

```
>> norm(u(:,1)-q(:,1),2)

ans =

    2.0000

>> norm(u(:,3)-q(:,3),2)

ans =

  2.0266e-015
```

SVD Decomposition need not be unique but the singular values must be same. The MATLAB function for svd could be using any other method to compute the matrices u and v but thing to note here is that u and v are same and that it is also similar in magnitude to the matrix q (matrix of eigen vectors), only the signs are inverted in some columns. This can be seen in the two graphs, in first one the comparison is between absolute values. In the next graph abs() is not used and normal comparison is made.




Graph comparing values of corresponding columns of matrices u and q
Norm values of corresponding vector difference
ith column of the matrix (i belongs to [1,30])
2.5
2
1.5
1
0.5
0
0
5
10
15
20
25
30