Iterative Methods to Compute Eigenvectors and Singular Vectors

Helen Dong

Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract

Eigenvectors and eigenvalues are important in applications of linear algebra. We will examine iterative ways such as the Power Method, the QR Method, and the Inverse Iteration Method to compute eigenvectors and singular vectors. These computational problems give rise to a several number of applications including, but not limited to, topics in mathematics, economics, and biology.

Introduction

At first glance, when asked to find eigenvalues, the typical approach is a naive method, which is to find the roots of the characteristic polynomial of the matrix. However, with larger and more complex matrices, this method is not feasible to find such eigenvalues and singular vectors. This is because of the numeric instability of the determinant function. The relation between the eigenvalue problem and polynomials displays that it is difficult to find eigenvalues and eigenvectors of a matrix larger than a $4 \times 4$. Thus, it requires one to look at iterative methods to find eigenvectors of these larger matrices. Although some of the iterative methods are simple to understand, they are not perfect and can only sometimes approximate the eigenvector. To understand how the iterative methods work, we will have to review some basic concepts of linear algebra.

Review of Topics

We begin by reviewing some basics of linear algebra. It is assumed that the reader is familiar with matrix and vector multiplication.

**Definition 1.1.** Suppose we have a matrix $A \in \mathbb{R}^{n \times n}$. A nonzero vector $x$ is an eigenvector of $A$ with a corresponding eigenvalue of $\lambda$ if $Ax = \lambda x$. We note that eigenvalues are scalar multiples of a vector.

**Definition 1.2.** The QR decomposition of a matrix $A$ is represented as $A = QR$, where $Q$ is an orthogonal matrix and $R$ is an upper triangular matrix. It is important to note that $Q^T Q = I$ and that the columns of $Q$ are orthogonal vectors that span the same column space as the columns of $A$.

**Definition 1.3.** The characteristic polynomial of $A$ is the degree $n$ polynomial defined by $P_A(x) = \det(Ix - A)$.

Mathematical Background and Algorithms of Computation

1 The Power Method

In the power method, we have a vector $v_0$ which could be either an approximation to the dominant/largest eigenvector or a random vector. The result generates a sequence of vectors $A^k v_0$, ...
where $\vec{v}_0$ is some original nonzero vector. We note that if $\vec{x}$ is an eigenvector for $A$, we have that $A\vec{x} = \lambda \vec{x}$ and $A^k \vec{x} = \lambda^k \vec{x}$. The sequence of vectors will eventually converge to a dominant eigenvector, associated with the largest eigenvalue.

Before beginning the power method, we need to give an order to the eigenvalues, i.e.

$$|\lambda_1| > |\lambda_2| > ... > |\lambda_n|$$

We let $w$ be an approximation of $A$ with $||w|| = 1$. We can write $w$ as a linear combination so we have that

$$w = c_1 q_1 + ... + c_n q_n$$

$$\Rightarrow A w = c_1 \lambda_1 q_1 + ... + c_n \lambda_n q_n$$

$$\Rightarrow A^k w = c_1 \lambda_1^k q_1 + ... + c_n \lambda_n^k q_n$$

We can factor our $\lambda_1$ so we have

$$\Rightarrow A^k w = \lambda_1 (c_1 q_1 + ... + c_n \left( \frac{\lambda_n}{\lambda_1} \right)^k q_n)$$

Since all the eigenvalues are supposedly real, distinct and ordered such that the first one has the greatest magnitude, second has the second greatest magnitude, and so on, we can conclude that for all $n \geq 2$, we can see convergence:

$$\lim_{k \to \infty} \left( \frac{\lambda_n}{\lambda_1} \right)^k = 0$$

So as $k$ increases, we can see that $A^k w$ will approach $c_1 \lambda_1^k q_1$ and for extremely large values of $k$, we get that

$$q_1 = \frac{A^k w}{||A^k w||}$$

At each iteration, the value of $w$ gets closer to the eigenvalue $q_1$. This algorithm is stopped when the approximation is close enough to the desired eigenvector.

To summarize:

We pick a starting vector $w$ with $||w|| = 1$. For $k \geq 1$, we let $v = A w^{k-1}$ and $w_k = \frac{v}{||v||}$

As an example, let us consider the matrix

$$\begin{pmatrix}
5 & -2 & -2 \\
-3 & 5 & 0 \\
23 & -19 & -6
\end{pmatrix}$$

We first begin with an initial guess of eigenvector

$$\begin{pmatrix}
1 \\
0 \\
0
\end{pmatrix}$$

It is important to note that this vector does indeed have a magnitude of 1, which is what we need to
begin the power method. Thus, we can begin the power method by multiplying it out.

\[
\begin{bmatrix}
5 & -2 & -2 \\
-3 & 5 & 0 \\
23 & -19 & -6
\end{bmatrix}
\begin{bmatrix}
1 \\
0 \\
0
\end{bmatrix}
= \begin{bmatrix}
5 \\
-3 \\
23
\end{bmatrix}
\approx \begin{bmatrix}
-15 \\
-30 \\
34
\end{bmatrix} \Rightarrow \begin{bmatrix}
5 & -2 & -2 \\
-3 & 5 & 0 \\
23 & -19 & -6
\end{bmatrix}
\begin{bmatrix}
-15 \\
-30 \\
34
\end{bmatrix}
= \begin{bmatrix}
-83 \\
-105 \\
21
\end{bmatrix} \approx \begin{bmatrix}
-83 \\
-105 \\
21
\end{bmatrix}
\]

After continuing this process, we can see that the values eventually converge to

\[
\begin{bmatrix}
4 \\
4 \\
2
\end{bmatrix}
\]
so then we can see that \( \begin{bmatrix}
2 \\
2 \\
1
\end{bmatrix} \) is approximately (but not exactly) the eigenvector and 2 is the largest eigenvalue.

If we wanted to find singular values and singular vectors, given a matrix \( A \), we would need to find \( AA^T \) and \( A^T A \) and follow the same method to find the singular values and singular vectors. It is important to note that when we find the eigenvalues of \( AA^T \) and \( A^T A \), we need to take the square root of those values to get the singular values.

A downfall of the power method is that it only returns one eigenvector estimate and it is only in correspondence to the eigenvalue with the largest magnitude. Also, the limit is only distinct when the eigenvalues are distinct; otherwise, we cannot conclude that \( \lim_{k \to \infty} \left( \frac{\lambda}{\lambda_n} \right)^k = 0 \). However, this method works well on large and sparse matrices when only a singular eigenvector is required. The power method is faster with sparse matrices because with more zeroes, there are less actual computations, which will speed up the process (zero times anything is zero). It is also more accurate because round-off errors will not have to be addressed. We want to note that this method also works better when the ratio between the largest and the second largest value is big. Furthermore, the power method is quick because it involves matrix-vector multiplication, which is faster than matrix-matrix multiplication.

2 The Inverse Iteration Method

We begin by noting that if \( A \) is invertible and has real, nonzero eigenvalues \( \{\lambda_1, \ldots, \lambda_n\} \), then we know that \( A^{-1} \) has eigenvalues \( \left\{ \frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_n} \right\} \). Then, if \( \lambda_1 > \lambda_2 > \ldots > \lambda_n \), we know that \( \frac{1}{\lambda_1} < \frac{1}{\lambda_2} < \ldots < \frac{1}{\lambda_n} \).

Thus, we can apply the power method on \( A^{-1} \), which will allow us to obtain the eigenvalue \( \lambda_n \) and eigenvector \( q_n \). The inverse iteration method works on the property that if \( A\tilde{x} = \lambda \tilde{x} \), then \( A^{-1} \tilde{x} = \lambda^{-1} \tilde{x} \). Applying the power method on such will allow us to find the eigenvalue that is the smallest in magnitude if we know \( A^{-1} \). However, if \( A^{-1} \) is not known, we can solve \( x = A^{-1} b \), which is equivalent to solving \( Ax = b \).

So we begin the inverse iteration method a similar way to the power method. We pick a starting vector \( w \) with \( ||w|| = 1 \). For \( k \geq 1 \), we solve \( Av = w^{k-1} \) for \( v \) and \( w_k = \frac{v}{||v||} \). The benefit of the
inverse iteration method is that it can be used to find any eigenvalue (with some adaptations) rather than just the extreme ones which can be achieved by the power method. Similar to the power method, the inverse iteration will also see convergence. It is important to note that this convergence can be close /one.pnum if the ratio of the first largest eigenvalue to the second largest eigenvalue of \((A - c_k I)^{-1}\) is close to 1.

For instance, for any \(z \in \mathbb{R}\), we know that \(B = A - zI\) has the eigenvalues \{\(\lambda_1 - z, ..., \lambda_n - z\}\). If we choose \(z\) carefully and make it be close to an eigenvalue \(\lambda_j\) of \(A\), we can see that \(\lambda_j - z\) is the smallest eigenvalue of \(B\) (in magnitude). Then we apply the inverse iteration shifted. So we pick a starting vector \(w\) with \(||w|| = 1\). For \(k \geq 1\), we solve \((A - zI)v = w^{k-1}\) for \(v\) and \(w_k = \frac{v}{||v||}\). This method (with the shift) will allow us to calculate any eigenvalue in the matrix. However, it must begin with an initial approximation with a magnitude of 1. It is important to choose the shift carefully because by choosing a shift close to the desired eigenvalue, one is able to see that the inverse iteration method will converge rather quickly.

This method is especially effective when a very close approximation to an eigenvalue (or singular value) is known and only its specific eigenvector (or singular vector) needs to be calculated. Thus, this method is more advantageous over the power method because it can find more than just the extreme eigenvectors and singular vectors. However, it requires a shift of \(A - c_k I\), which makes this method unattractive when that factorization is expensive. Similar to the power method, the inverse iteration method is quicker because it involves matrix-vector multiplication, which is faster than matrix-matrix multiplication. Furthermore, the inverse iteration method works better with sparse matrices because it makes the matrix-vector multiplication simpler (because it will be just zeroes).

If we want to find singular vectors and singular values, we will want to apply the same process to \(AA^T\) and \(A^T A\) but when finding the singular values, we find the eigenvalues and take the square root.

### 3 The QR Method

We want to recall that when we decompose \(A\) into \(QR\), that \(Q\) has orthonormal columns and \(R\) is upper triangular.

Given a matrix \(A\), to find singular values, we just have to find the eigenvalues of \(AA^T\) and \(A^T A\) and take the square root of those values. Below, we will learn the QR Method in order to find singular values. It will be described for a general matrix \(A\) but it is important to keep in mind that \(A\) can be either \(A^T A\) or \(AA^T\) and it is also a square matrix. Suppose \(A_1 = RQ\). We know that \(A\) and \(A_1\) has the same eigenvalues because \(A_1\) is similar to \(A = Q^{-1}AQ\). So we have

\[
QR\bar{x} = \lambda\bar{x} \implies RQ(Q^{-1}\bar{x}) = \lambda(Q^{-1}\bar{x})
\]

Then we factor out this new matrix (in this case \(A_1\)) into \(Q_1 \text{ and } R_1\). Then we reverse \(A_1 = Q_1R_1\) to \(R_1Q_1\), which will give us \(A_2\). This will still not give us a change in eigenvalues based on our previous argument. Eventually, luckily enough, the eigenvalues begin to appear on the diagonal. After enough iterations, if we have \(A\) to be an \(n \times n\) matrix, at entry \(A[n, n]\), the accurate eigenvalue will appear. So we can restart this method and do the same thing eliminating the last row and column.

There are other ways to make this method successful. For instance, one way is to shift the matrix by
a multiple of $I$. Thus we factor $A_k = c_k I$ into $Q_k R_k$ so we have

$$A_{k+1} = R_k Q_k + c_k I$$

and $A_{k+1}$ has the same eigenvalues as $A_k$, and the same as the $A_0$, which is the original $A$.

Another way is to obtain off-diagonal zeroes before beginning this method. This will make our process simpler because one of the goals of the QR Method is to obtain zeroes on the off diagonals from switching $Q$ and $R$.

After getting the eigenvalues, we can multiply all the $Q$'s together to get the matrix of eigenvectors (or singular vector).

As an example, suppose we have a matrix

$$A = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
16 & 8 & 4 & 2 & 1 \\
81 & 27 & 9 & 3 & 1 \\
256 & 64 & 16 & 4 & 1 \\
625 & 125 & 25 & 5 & 1
\end{bmatrix}$$

At first sight, we see that it is a large matrix and using a non-iterative method could be atrocious. So we decompose this matrix is QR. So we begin the QR Method and we can find

$$A_1 = Q_1^T A Q_1$$

As an example, suppose we have a matrix

$$A_1 = \begin{bmatrix}
11.7037 & 95.9499 & -391.8382 & 548.762 & 141.9901 \\
3.7455 & 11.4289 & -12.0675 & -4.4129 & -6.0307 \\
-1.6043 & -1.3135 & -0.192 & 0.1581 & -0.4721 \\
0.6423 & 0.0007 & 0.0670 & 0.0554 & -0.0627 \\
0.0580 & -0.0221 & -0.0096 & -0.0051 & 0.004
\end{bmatrix}$$

Then we find the QR decomposition of this matrix. After several iterations, we get

$$A = \begin{bmatrix}
45.9697 & 333.0696 & -514.755 & 307.7187 & -66.7932 \\
-0.0021 & -28.953 & 62.9322 & -47.4590 & 12.6896 \\
-0.00 & -0.00 & 6.7954 & -9.3745 & 3.5876 \\
-0.00 & -0.00 & 0.00 & -0.8496 & 0.6473 \\
-0.00 & -0.00 & 0.00 & -0.00 & 0.0375
\end{bmatrix}$$

After many iterations, we can see that the eigenvalues begin to show up on the diagonal. It is true that the eigenvalues for the matrix $A$ are 45.9697, -28.953, 6.7954, -0.8496, 0.0375, which are exactly the values on the diagonal. If we keep track of the $Q$'s, we can see that when we multiply them together, we get the eigenvectors:

$$\begin{bmatrix}
-0.0352912 & -0.0437719 & 0.052657 & 0.0335 & 0.0108 \\
-0.0814649 & -0.031881 & -0.1498 & -0.2283 & -0.1245 \\
-0.195823 & 0.0643936 & -0.3253 & 0.22621 & 0.4908 \\
-0.435072 & 0.342399 & -0.14108 & 0.6385 & -0.7682 \\
-0.874348 & 0.93578 & 0.9214 & -0.698 & 0.3914
\end{bmatrix}$$
Pseudo code for QR Method

Create a function to create a matrix.
    for i from 1 to the end rows
    for i from 1 to the end cols
        Mat[i,j] = input from 1d array
Create a function to find the eigenvalues. Given a matrix $A$, qr decompose it into $Q_1$ and $R_1$. Find a newA that is equal to $R_1 Q_1$.
    for (iterate through 1000 times)
        decompose the newA into Q and R.
        find new Q and R.
        reassign newA to be the newR * newQ
Create a function to find all the eigenvalues.
Call the function to find eigenvalues on a smaller matrix $A$.
Add all the eigenvalues to a 1d array.
Return the eigenvalue array.

To find the singular values, given a matrix $A$, we pass in $A^T A$ and $AA^T$ to find those eigenvalues. Then we take the square root of each to find the singular values.

To find the eigenvectors, we keep track of the Q's and multiply them and add it to a 2d array and return it at the end.
Code for QR Method

December 4, 2019

In [2]: using LinearAlgebra
   function findAAT(A)
       return A*transpose(A)
   end

   function findATA(A)
       return transpose(A) * A
   end

   function createMatrix(row, cols, input)
       Mat = zeros(Int64, rows, cols)
       for i = 1:rows
           for j = 1:cols
               Mat[i,j] = input[(i-1)*cols+j]
           end
       end
       return Mat
   end

Out[2]: createMatrix (generic function with 1 method)

In [45]: function singularVal(A, row)
       decomp1 = qr(A)
       Q1 = decomp1.Q
       R1 = decomp1.R
       newA = R1*Q1
       evec = Q1 * (qr(newA).Q)
       for i = 1:1000
           decompx = qr(newA)
           Qx = decompx.Q
           Rx = decompx.R
           newA = Rx*Qx
           evec = evec * Qx
       end
       return (newA, newA[row,row], evec)
   end

Out[45]: singularVal (generic function with 1 method)
In [46]: function otherEvals(A, size)
    
    values = zeros(Float64, size)
    val = size
    for i = 1:size
        value = singularVal(A, size)
        A = value[i][1:(val-1), 1:(val-1)]
        size = size - 1
        values[i] = value[2]
    end
    return values
end

Out[46]: otherEvals (generic function with 1 method)

In [47]: println("enter how many rows")
    
    rows = parse(Int, readline());
    println("enter how many cols")
    cols = parse(Int, readline());
    println("enter " * string(rows*cols) * " values for the matrix")
    input = readline()
    n = split(input, " ")
    Adj = createMatrix(rows, cols, parse(Int, n))
    AAT = findAAT(Adj)
    ATA = findATA(Adj)
    println(ATA)
    println(AAT)

    enter how many rows
    stdin> 5
    enter how many cols
    stdin> 5
    enter 25 values for the matrix
    stdin> 1 2 3 4 5 43 43 32 21 14 65 65 43 24 74 89 78 67 89 90 67 89 78 67 56
    [18485 18981 15363 14877 17179; 18981 20083 16345 15376 17426; 15363 16345 13455 12905 14043; 14305 13905 12505 12005 11505; 55 379 790 1252 1027; 379 5359 8506 12454 11395; 790 8506 16351 22532 19246; 1252 12454 22532 3

In [48]: #evals of AAT (take square root to find singular values of A)
    
    otherEvals(AAT, rows)

Out[48]: 5-element Array{Float64,1}:
    0.15089938034332143
    220.1420858469444
    1091.4210720753226
    1454.5770212245313
    79632.7089214728

In [50]: #evecs of AAT
    
    U = singularVal(AAT, rows)[3]
Out[50]: SE5 Array<Float64, 2>:
    0.0183323  0.0091261  -0.0363003  0.178904  -0.982984
    0.278979   0.350857    0.300908   -0.88403  -0.165405
    0.44309    -0.799497   0.383363   0.131865   0.010698
    0.619913   -0.0430034  -0.778837  -0.0815048  0.02509
    0.584135   0.579754    0.39317    0.403049   0.0751043

In [51]: eigen(AAT)

Out[51]: Eigen<Float64, Float64, Array<Float64, 2>, Array<Float64, 1>}
eigenvalues:
    5-element Array<Float64, 1>:
        0.1508993803434848
        220.14208584694842
        1091.4210720753213
        1454.5770212245302
        79632.70892147286
eigenvectors:
    SE5 Array<Float64, 2>:
        0.98387    0.159209    0.0765001   -0.0163445   -0.0231014
        0.161181   -0.766601   -0.489315    0.293423    -0.246612
       -0.0105165  0.148597    -0.46725    -0.753376   -0.438075
       -0.0258729  -0.279833    0.695226   -0.106324   -0.652978
       -0.0723962  0.535339   -0.230339    0.57858    -0.566003

In [61]: ##evals of ATA (take square root to find singular values of A)
otherEvals(ATA, cols)

Out[61]: 5-element Array<Float64, 1>:
        0.15089938034315725
        220.14208584693353
        1091.421072075192
        1454.577021224531
        79632.7089214729

In [62]: ##reversing for final calc
    temp = reverse(otherEvals(ATA, cols))
    S = zeros(Float64, rows, cols)
    for i = 1:min(rows, cols)
        S[i, i] = sqrt(abs(temp[i]))
    end

In [63]: ##svecs of ATA
    V = singularVal(ATA, cols)[3]

Out[63]: SE5 Array<Float64, 2>:
        0.473847  -0.021461    0.251457    -0.820969   -0.194391
        0.497238    0.04993    0.513616     0.293155    0.632866
        0.409654    0.267565    0.196534     0.458956   -0.71507
        0.402827    0.557699   -0.689912   -0.0456264   0.22055
        0.445126   -0.783854   -0.397949     0.165372   -0.0415292
In [64]: eigen(ATA)

Out[64]: Eigen{Float64,Float64,Array{Float64,2},Array{Float64,1}}
eigenvalues:
5-element Array{Float64,1}:
    0.1508993803430506  
    220.142085846944  
    1091.4210720753094  
    1454.5770212245156  
    79632.70892147289  
eigenvectors:
5×5 Array{Float64,2}:
    0.200358  0.82113  -0.148106  0.185285  -0.478891  
   -0.634341 -0.190843 -0.530665 -0.1787  -0.497645  
    0.712538 -0.360156 -0.209061 -0.392011 -0.406445  
   -0.219738  0.0628748  0.764568  -0.454065  -0.396262  
    0.0383707 -0.394536  0.261131  0.757554 -0.446097  

In [65]: V = transpose(V)
    
U*S*V

Out[65]: 5×5 Array{Float64,2}:
    0.0373292  2.51024  3.46786  2.89979  2.96239  
    50.4609  40.6528  29.7696  28.6475  24.4106  
    61.4807  67.7318  46.4475  24.5368  74.8417  
    77.4484  73.3386  65.6051  87.3627  89.1925  
    75.9844  91.511  78.7196  69.5052  51.8608
Discussion and Results

From the code, the advantages and disadvantages of the QR method arise and become clear. Although the QR method is a simple, redundant algorithm, it has some disadvantages. For example, it suffers problems with multiple eigenvalues and eigenvalues with equal modulus. Furthermore, the QR method suffers drawbacks because the complexity of one step in the QR algorithm runs in $O(n^3)$ time because multiplying matrices is expensive. Also, several steps are required before a convergence can be seen so the QR method can be arbitrarily slow in many instances. The QR Method can produce inaccurate results, but if the matrix is sparse initially, it works more efficiently and will produce more accurate results.

Also, depending on how many times the method is iterated through, the accuracy of the eigenvalues can change. For instance, some matrices require more than 100 iterations while others produce a very accurate value with just 100 iterations.

Despite the amount of iterations, for larger matrices, the accuracy will never be perfect. I've noticed that even with 1000 iterations, the eigenvalues are found almost perfectly, while the eigenvectors have percent errors. This indicates then when we find $U$ and $V$ from $AA^T$ and $A^T A$ respectively, the eigenvectors will be off slightly. However, because of this, when we find the SVD $U\Sigma V^T$, it will not return the original $A$ exactly, but it will produce an $A$ will entries very close to the original $A$. Also we note that the code produces eigenvectors in negative form but that is still correct because eigenvectors can be multiplied by a scalar multiple. For instance, using the eigen function, it may claim the eigenvectors are
\[
\begin{pmatrix}
0.03 \\
-0.0098 \\
0.034 \\
0.005
\end{pmatrix}
\]
but instead, the code will produce
\[
\begin{pmatrix}
-0.03 \\
0.0098 \\
-0.034 \\
-0.005
\end{pmatrix}
\], which is just the eigenvector given multiplied by a constant of -1.

Conclusion

From these methods, we can see that iterative methods can be quite useful in finding eigenvalues and singular values of large matrices. Since the determinant function is numerically unstable, the iterative methods prove to be more useful with larger matrices. Though these iterative methods have seemingly large benefits, they are not always accurate and can have round-off errors. Also, they can be excessively slow in computation. Overall, they are useful in computing eigenvectors and singular vectors in larger matrices.

References

The QR Algorithm for Finding Eigenvectors, Eric Mikida, 2011
Strang Chapter 11.3
Foundations of Data Science by Avrim Blum, John Hopcroft, and Ravindran Kannan, Chapter 3.7
Computational Aspects of F. L. Bauer’s Simultaneous Iteration Method, Heinz Rutishauser
QR Algorithm, Elias Jarlebring, 2014
The QR Algorithm, Hannes Thiel