MATH0058

FWMS2

BOOKWORK QUESTIONS

Basic power method (BPM). This finds the eigenvector associated with the eigenvalue of largest magnitude of a square matrix, called the dominant eigenvector. Let $A \in \mathbb{C}^{n\times n}$, and let each eigenvalue of A and its associated eigenvector be λ_i and v_i respectively where $i \in \{1, ..., n\}$. We must order the eigenvalues as so:

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

To implement the algorithm we choose a vector $v \in \mathbb{C}^n$. We iterate to obtain the sequence:

$$v, Av, A^2v, A^3v, ...$$

To avoid the sequence blowing up we re-scale the vectors at each stage by dividing through by its Euclidean norm, for example. As Watkins suggests, this choice of the Euclidean norm is not the only way to scale and we could divide through by the dominant eigenvalue to obtain a similar result [5]. The important thing is the eigenvector's direction [5]. Note that convergence means convergence of components of one vector to components of another.

Intuitive Explanation. I consider $\mathbb{R}^2$ for intuition and arbitrary $A = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$, $v = \begin{pmatrix} -10 \\ 40 \end{pmatrix}$

LISTING 1. Example of convergence in $\mathbb{R}^2$

```
import numpy as np
import matplotlib.pyplot as plt
A=np.array([[1, 1],[0,2]])
v= np.array([-10,40])
for i in range(50):
    plt.arrow(0,0,v[0],v[1],color=(0.01 + (i/60),0,1-i/60))
    print(v)
    v = A.dot(v)
    v = v/np.linalg.norm(v)
plt.show()
```

The Python code given in listing 1 results in Figure 1. The sequence converges to a multiple of the dominant eigenvector of A, namely to $\alpha \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $\alpha \approx 0.707$. Assume $A \in \mathbb{R}^{2\times 2}$ is diagonalisable. To keep ensure convergence, divide by the dominant eigenvalue at each step.

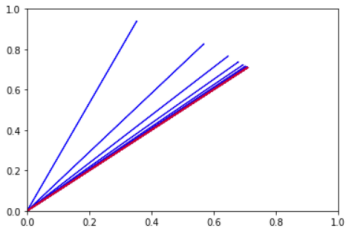

FIGURE 1. Convergence

$\frac{1}{\lambda_1^n} A^n = P \begin{pmatrix} 1 & 0 \\ 0 & \left(\frac{\lambda_2}{\lambda_1}\right)^n \end{pmatrix} P^{-1}$. The determinant of both sides is equal to $\left(\frac{\lambda_2}{\lambda_1}\right)^n$ and tends to zero as $n \to \infty$ therefore under iteration space collapses onto

$$\begin{pmatrix} x \\ y \end{pmatrix} = t \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad t \in \mathbb{R}$$

Rigorous explanation. Consider the case that our initial v happens to be an eigenvector. We get the sequence

$$v, \lambda v, \lambda^2 v, ...$$

Which when we re-scale by dividing through by λ at each step becomes:

$$v, v, v, ...$$

This converges to v but v needn't be the dominant eigenvector. Let's consider the case where v isn't an eigenvector. Assume that A has n linearly independent eigenvectors, $v_1, ..., v_n$ and with each eigenvector is the eigenvalue, $\lambda_1, ..., \lambda_n$. The eigenvalues and eigenvectors must be ordered:

$$|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$$

This is for two reasons:

(1) If the largest absolute value of the eigenvalues is shared by two eigenvectors. then the sequence converges only if v is an eigenvector.
(2) Convenience of notation.

We can write v as $v = a_1v_1 + a_2v_2 + \ldots + a_nv_n$. Re-scaling by dividing by λ_1 at each step, after m iterations we can express w_m, the vector obtained, as $w_m = \frac{1}{\lambda_1^m}A^m v = a_1v_1 + a_2(\lambda_2/\lambda_1)^m v_2 + \ldots + (\lambda_n/\lambda_1)^m a_n v_n$. Since $|\lambda_k/\lambda_1| < 1$ for all $k \neq 1$, we have that as $w_m \to a_1v_1$ as $m \to \infty$.

Subspace iteration. We are lead to consider $\langle A^m v \rangle$. This is because the BPM considers what Watkins terms "representatives" of this subspace, and as is very often the case, we want a broader more general approach[5]. The representative that we get is determined by how we scale. We generalise the BPM by considering a k-dimensional subspace of $\mathbb{C}^n$. The BPM considers $k = 1$. Watkins defines a metric $d(S, T)$, where S and T are k-dimensional subspaces of $\mathbb{C}^n$ and states that this metric is "the sine of the largest canonical angle between S and T" [5]. Define S, T, U as Watkins does. Then Theorem 2.1 in his text is equivalent to saying there exists a constant $C \in \mathbb{R}$ such that the largest canonical angle between $A^m S$ and T is less than or equal to $C|\lambda_{k+1}/\lambda_k|^m$ for all m. This implies $A^m S$ converges linearly to T with ratio and most $|\lambda_{k+1}/\lambda_k|$. Watkins argues this theorem's validity by considering $v \in S$, $v \neq 0 \in \mathbb{C}^n$ as the sum of two vectors: one in T and one in U. He notes that as $v \notin U$, the component of v that lies in T must be non-zero. Giving:

$$\begin{aligned} \frac{A^m v}{(\lambda_k)^m} &= a_1(\lambda_1/\lambda_k)^m v_1 + \ldots + a_k v_k && \text{Gets more significant} \\ &+ a_{k+1}(\lambda_{k+1}/\lambda_k)^m v_{k+1} + \ldots a_n(\lambda_n/\lambda_k)^m v_n && \text{Tends to 0} \end{aligned}$$

This gives that $v \in T$ will tend to a vector in T. The rate is linear with the given rate because every time we apply the algorithm, the coefficients of the component of the vector in U decrease. Specifically the smallest reduction in magnitude is seen at the $k + 1^{th}$ term which is scaled down by $|\lambda_{k+1}/\lambda_k|$.

The role of orthogonalisation and shifts in simultaneous iteration. It is advantageous to use orthogonal bases to avoid small numerical errors causing problems in a practical setting. If two vectors are orthogonal then they are more easily distinguished than if they were pointing in roughly the same direction, something orthogonalisation avoids. Secondly, because each subspace S_i is preserved under orthonormalisation, we get subspaces, $A^m S_i$ of dimensions up to $k - 1$ "at no extra cost" compared to finding the subspaces up to dimension k [5]. We have the appearance of block triangular matrices, and often eventually upper triangular matrices with the eigenvalues appearing on the diagonal. These matrices are less computationally expensive because they allow for significant reduction in the number of operations due to the quantity of 0 entries. Shifts are needed to make the algorithm useful and 'competitive' in practical settings due to the increased rate of convergence (ROC) [5]. Shifting the diagonals of the matrix by σ allows the ROC to become $|(\lambda_{k+1} - \sigma/(\lambda_k - \sigma)|$ and hence appropriate selection of σ forces ROC to improve. We may

change our value of σ at each iteration so that "convergence is better than linear", though this is numerically unstable [5].

Moderate questions

The QR algorithm and its relationship to simultaneous iteration. The QR algorithm finds eigenvalues of a matrix through iterative decomposition into a unitary matrix, Q, and an upper triangular matrix, R. The code below will print a result of the QR algorithm after 1000 iterations.

Listing 2. QR iteration

```
import numpy as np

A= np.random.randn(3,3)
for i in range(1000):
    decomp = np.linalg.qr(A)
    A = decomp[1]@decomp[0]
print(A)
```

The given matrix is decomposed to the form $A = QR$. Then we take the Q and the R and compute the matrix RQ. We then compute the QR decomposition of the matrix RQ. We repeat this for as long as desired. The output is a matrix with the approximate eigenvalues of our input matrix along the diagonal and values of ever decreasing magnitude below it. In the limit we obtain an upper diagonal matrix with the eigenvalues of our input matrix along the leading diagonal. We may chose to stop when one of the elements below the diagonal reaches a certain value that is arbitrarily close to 0. In Listing 2, a 3×3 matrix is used but a square matrix of any dimension is a valid input.
Now I shall explain the link between the QR algorithm and simultaneous iteration. Watkins recommends to perform subspace iteration as so [5]:

- Given an orthonormal basis of $A^m S$, $\{q_1^m, ..., q_k^m\}$, calculate $Aq_1^m, ...Aq_k^m$.
- Orthonormalise these vectors obtained to get an orthonormal basis for $A^{m+1}S$.

The QR algorithm also involves finding orthonormal bases. Watkins states that the algorithm is the Gram-Schmidt process using matrices to perform the operations leading to orthonormalisation [5]. In QR decomposition, we are provided with an orthonormal basis for $A^m S$ at each step, just like the simultaneous iteration. Note the following:

$$A_0 = Q_0R_0 \implies A_1 = R_0Q_0 \implies Q_1R_1 = R_0Q_0 \implies R_1 = Q_1^{-1}R_0Q_0$$
$$\implies R_1 = Q_1^{-1}Q_0^{-1}A_0Q_0 \implies R_1Q_1 = Q_1^*Q_0^*A_0Q_0Q_1 \implies A_2 = P^*A_0P$$

The above is generalised in Watkin's paper to reveal that the algorithm results in changing bases to get orthogonal bases at each step, by the change of basis formula. This is equivalent to subspace iteration. The QR algorithm is then a sequence of similarity transformations because inductively we have that A_m is A_0 multiplied on the left by $Q_{m-1}^*...Q_0^* = (Q_0...Q_{m-1})^*$ and multiplied on the right by $Q_0...Q_{m-1}$ which is the similarity transformation $A_m = P_m^{-1}A_0P_m$ because the Q_i's are unitary matrices and so their complex transposes are their inverses.

Hessenberg form and shifts of origin. Shifts of origin involve replacing the matrix obtained after m iterations with a matrix $A_m - \sigma_m I$. This increases the ROC and we can easily obtain the eigenvalues of A by reversing the shift. THe ROC increases because, instead of it being $|\lambda_n/\lambda_{n-1}|$, we force it to be $|(\lambda_n - \sigma_m)/(\lambda_{n-1} - \sigma_m)|$. If we use $\sigma_m = a_{nn}^m$ then we get at least quadratic convergence because of how close the eigenvalues are to our shifts and the fact that this gets improves under each iteration. An even better choice is to take σ_m to be the solution, t, that is closest to a_m^m, of

$$t^2 - (a_{n-1,m-1}^m + a_{n,n}^m)t + a_{n-1,m-1}^m a_{n,n}^m - a_{n-1,n}^m a_{n,n-1}^m$$

As Strang notes at 17:50 in his lecture, the computationally expensive part of this method is in the computation of the QR decomposition and that an effective way to further reduce the cost of the

algorithm is to use Hessenberg matrices [4]. A matrix is in Hessenberg form if it has entries of only zeros below the subdiagonal. This form is "preserved under QR iteration" and the amount of zeros reduces computation time significantly because we know to ignore certain entries hence reduce the computational cost of the algorithm [5].

Advanced questions

The QR algorithm and inverse iteration. Inverse iteration is useful when wanting to find non-dominant eigenvectors, and when we apply inverse iteration to a non-eigenvector we obtain the eigenvector associated with the eigenvalue of smallest magnitude. Equation (4.2) in Watkins' paper is the same as equation (3.5) in his paper and likewise for (4.3) and (3.7), but A's are swapped for $(A^*)^{-1}$'s and R's are swapped for L's. Indeed Watkins gives that $L_m = (R_m^*)^{-1}$ and that $\hat{L}_m = (\hat{R}_m^*)^{-1}$. The similarity lies in the fact that $A^m S$ and $A^{-m} S$ are orthogonal complements, that is we're considering the dual problem and because spaces and matrices are orthogonal we may consider conjugate transposes. Transposes of upper diagonal matrices are lower diagonal and vice-versa. It is also noteworthy that, for compatible matrices A, B, $(AB)^* = B^* A^*$. Hence, in a sense, we're considering the same problem but just with respect to transposes and orthogonal spaces.

LAPACK. Non-symmetric eigenvalue problems are solved with LAPACK by using the 'Schur Factorisation'. The specifics of this decomposition correspond to a QR decomposition with a Hessenberg matrix but allows the k leading diagonal entries to be occupied by any k eigenvalues [1]. The drivers can compute eigenvalues, left and right eigenvectors, and also the condition numbers of the eigenvalues and eigenvectors. To obtain eigenvectors, the subroutine 'xHSEIN' performs inverse iteration on the Hessenberg matrix and then another subroutine, 'xORMHR' multiplies these vectors by an orthogonal matrix, Q such that $A = QHQ^H$ in order to obtain the eigenvalues of the input matrix [2]. In general the methods used by LAPACK to solve eigenvalue problems can be reduced to three steps [3].

(1) Use of orthogonal transformations to a more manageable, efficient form
(2) Solving the problem for the more manageable matrix
(3) Adjusting the solution to step (2) to make it work for the original input matrix

The advantage of doing this is that computation time is reduced by handling simpler matrices over more dense ones and that the use of orthogonal matrices reduces computational error. However these processes are more computationally expensive comparitatvely when considering smaller dimensional matrices [3].

References

[1] Susan Blackford. 1999. URL: https://www.netlib.org/lapack/lug/node31.html.
[2] Susan Blackford. 1999. URL: https://www.netlib.org/lapack/lug/node50.html.
[3] Susan Blackford. 1999. URL: https://www.netlib.org/lapack/lug/node70.html.
[4] Gilbert Strang. *12. Computing Eigenvalues and Singular Values*. 2019. URL: https://www.youtube.com/watch?v=d32WV1rKoVk&.
[5] David S. Watkins. "Understanding the QR Algorithm". In: *SIAM Review* 24.4 (1982), pp. 427–440. ISSN: 0036-1445.