Yingbo Hua

Asymptotical Orthonormalization of Subspace Matrices Without Square Root

Subspace computation is fundamental for many signal processing applications. A well-known tool for computing the principal subspace of a data matrix is the power method. During the iterations of the power method, a proper normalization is essential to avoid numerical overflow or underflow. Normalization is also needed to achieve desirable properties such as orthonormalized subspace matrices. We review a number of normalization techniques for the power method, which include the conventional as well as nonconventional ones. In particular, we introduce a new method of normalization to achieve asymptotical orthonormalization of subspace matrices without the use of square root. This method is among a class of normalization methods that allow a simple adaptive implementation of the power method for subspace tracking.

Principal Subspace and the Power Method

Let a sequence of data vectors be described by the matrix: $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2 \cdots \mathbf{x}_m] \in C^{n\times m}$. The singular value decomposition (SVD) of the matrix $\mathbf{X}$ is denoted by

$$\mathbf{X} = \sum_{i=1}^{\min(m,n)} \sigma_i \mathbf{u}_i \mathbf{v}_i^H$$

where the conventional notations are adopted and the singular values are in descending order [1]. We are interested in finding a rank-r principal subspace matrix $\mathbf{S} \in C^{n\times r}$ such that $\text{range}(\mathbf{S}) = \text{range}(\mathbf{U})$ where $\mathbf{U} = [\mathbf{u}_1\ \mathbf{u}_2 \ldots \mathbf{u}_r]$. If the SVD of $\mathbf{X}$ is given, the problem is solved of course. Yet, the complexity of SVD is in the order of $O(n^2 m)$ flops assuming $n < m$. If $r \ll \min(m, n)$, more efficient methods are available and desirable.

If a subspace matrix $\mathbf{S}$ is found and orthonormalized (i.e., $\mathbf{S}^H\mathbf{S} = \mathbf{I}$), then there is a unitary matrix $\mathbf{T} \in C^{r\times r}$ such that $\mathbf{S} = \mathbf{U}\mathbf{T}$. The matrix $\mathbf{T}$ and the principal singular values can be computed from the SVD of the smaller matrix: $\mathbf{S}^H\mathbf{X} \in C^{r\times m}$ as $\mathbf{S}^H\mathbf{X} = \mathbf{T}^H\Sigma\mathbf{V}^H$ where $\Sigma = \text{diag}(\sigma_1\ \sigma_2 \cdots \sigma_r)$ and $\mathbf{V} = [\mathbf{v}_1\ \mathbf{v}_2 \cdots \mathbf{v}_r]$.

For some applications, a subspace matrix $\mathbf{S}$ is equally sufficient as the matrix $\mathbf{U}$. Therefore, computing $\mathbf{S}$ can be treated as a separate problem as is the case in this letter.

There are many variations of the power method for computing a subspace matrix $\mathbf{S}$. However, most of these variations have the following procedure in common:

For $k = 0, 1, 2, \ldots$, until $\mathbf{S}_k$ converges, do:

$$\begin{aligned}
&\hat{\mathbf{T}}_{k+1} = \mathbf{X}^H\mathbf{S}_k; && nmr \text{ flops;}\\
&\hat{\mathbf{S}}_{k+1} = \mathbf{X}\hat{\mathbf{T}}_{k+1}; && nmr \text{ flops;}\\
&\mathbf{S}_{k+1} = f(\hat{\mathbf{S}}_{k+1}, \mathbf{S}_k); && O(nr^2) \text{ flops;}
\end{aligned}$$

End.

With an initial matrix $\mathbf{S}_0$ (satisfying a weak condition shown later) and the assumption $\sigma_r > \sigma_{r+1}$, all power methods converge to a desired subspace matrix $\mathbf{S}$, i.e., $(\text{range}S_\infty = \text{range}(\mathbf{S}) = \text{range}(\mathbf{U})$, and have an asymptotical error in the order of $(\sigma_{r+1}/\sigma_r)^{2k}$ where k is the iteration index. The simplicity of the power method makes it an attractive choice for adaptive computation of subspace. In an adaptive power algorithm, the data matrix $\mathbf{X}$ may be updated after each iteration of the algorithm. An adaptive algorithm can provide a good tradeoff between accuracy and complexity. With proper approximations in the algorithm design, a complexity of $O(nr)$ flops per iteration can be achieved. A recent design and analysis of adaptive algorithms are available in [3].

However, an adaptive implementation of the power method is affected significantly by the normalization step: $\mathbf{S}_{k+1} = f(\hat{\mathbf{S}}_{k+1}, \mathbf{S}_k)$. In this note, we will not address the full impact of the normalization step on adaptive implementations, but rather discuss several interesting choices of the normalization function $f(\hat{\mathbf{S}}_{k+1}, \mathbf{S}_k)$.

We note that if $r = 1$ is chosen, $\mathbf{S}_k$ converges to the principal vector $\mathbf{u}_1$. With a deflation on the data matrix, i.e., replacing $\mathbf{X}$ by $(\mathbf{I} - \mathbf{u}_1\mathbf{u}_1^H)\mathbf{X}$, the remaining principal vectors can be obtained in a similar fashion. We focus on the general case $r \geq 1$. This case is also known as simultaneous iteration where the dominant principal vectors can be further retrieved with a higher accuracy [2]. Some unscrambling procedure is required to achieve that but will not be addressed here.

For simple expressions of flop counts, we will assume $n \gg r \gg 1$. Each flop is a complex multiplication. Additions are neglected

1053-5888/04/$20.00©2004IEEE

although the number of additions is generally about the same as the number of multiplications.

Normalization Methods

Method 1: Using QR Decomposition

QR decomposition is perhaps the most conventional method of normalization. Here, the function $f(\hat{\mathbf{S}}_{k+1}, \mathbf{S}_k)$ is performed via a thin QR decomposition:

$$\begin{cases} \hat{\mathbf{S}}_{k+1} \Rightarrow \mathbf{Q}_{k+1}\mathbf{R}_{k+1} \\ \mathbf{S}_{k+1} = \text{first\_}r\text{\_columns\_of\_}\mathbf{Q}_{k+1} \end{cases}$$

where $\mathbf{R}_{k+1}$ is a tall $n \times r$ upper-triangular matrix and $\mathbf{Q}_{k+1}$ is a $n \times n$ unitary matrix. There are a variety of algorithms available for QR decomposition, which include Householder transformation, Givens transformation, and Gram-Schmidt transformation [1].

Householder Transformation

The Householder transformation has the form $\mathbf{H} = \mathbf{I} - 2(\mathbf{h}\mathbf{h}^H/\|\mathbf{h}^2\|)$ where $\|\mathbf{h}\|$ is the 2-norm of $\mathbf{h}$. To achieve $\mathbf{y} = \mathbf{H}\mathbf{x}$ with $\|\mathbf{x}\| = \|\mathbf{y}\|$, we can simply choose $\mathbf{h} = \mathbf{x} - \mathbf{y}$. A unique property of the Householder transform is illustrated in Figure 1. To transform a vector $\mathbf{x}$ by $\mathbf{H}$ into such a vector $\|\mathbf{x}\|\mathbf{e}$ where $\mathbf{e} = [10\ldots0]^T$, we hence choose $\mathbf{h} = \mathbf{x} - \|\mathbf{x}\|\mathbf{e}$. To transform a vector $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$ by $\mathbf{H}$ into another vector $\mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}$ where $\mathbf{x}_1 = \mathbf{y}_1$ and $\|\mathbf{x}_2\| = \|\mathbf{y}_2\|$, we should define $\mathbf{h} = \begin{bmatrix} \mathbf{0} \\ \mathbf{x}_2 - \mathbf{y}_2 \end{bmatrix}$.

To compute the QR decomposition $\hat{\mathbf{S}}_{k+1} \Rightarrow \mathbf{Q}_{k+1}\mathbf{R}_{k+1}$, a sequence of r Householder transformations $\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_r$ can be constructed and applied. We should choose $\mathbf{H}_1$ such that all but the first element of the first column of $\mathbf{H}_1\hat{\mathbf{S}}_{k+1}$ are zero. We choose $\mathbf{H}_2$ such that all but the first two elements of the second column $\mathbf{H}_2\mathbf{H}_1\hat{\mathbf{S}}_{k+1}$ are zero. We repeat the process such that $\mathbf{H}_r \cdots \mathbf{H}_2\mathbf{H}_1\hat{\mathbf{S}}_{k+1} = \mathbf{R}_{k+1}$ is upper triangular.

For each $i = 1, 2, \ldots, r$, we may write $\mathbf{H}_i = \mathbf{I} - 2\mathbf{h}_i'\mathbf{h}_i^H$ where $\mathbf{h}_i$ requires $n - i + 1$ flops and one square root, and $\mathbf{h}_i' = \mathbf{h}_i/\|\mathbf{h}_i\|^2$ requires additional $2(n - i + 1)$ flops. To construct $\mathbf{h}_i$ and $\mathbf{h}_i'$ for all $i = 1, 2, \ldots, r$, we need about $3nr$ flops.

Given $\mathbf{h}_i$ and $\mathbf{h}_i'$ for all $i = 1, 2, \ldots, r$, the multiplication $\mathbf{H}_1\hat{\mathbf{S}}_{k+1}$ requires $2nr$ flops, the multiplication $\mathbf{H}_2(\mathbf{H}_1\hat{\mathbf{S}}_{k+1})$ requires an additional $2n(r-1)$ flops, and so on. (The multiplication by the factor 2 is not counted, which is equivalent to one shift of binary register.) To complete the computation $\mathbf{H}_r \cdots \mathbf{H}_2\mathbf{H}_1\hat{\mathbf{S}}_{k+1} = \mathbf{R}_{k+1}$, we need about nr^2 flops.

To obtain $\mathbf{S}_{k+1}$, we need to compute the left r columns of $\mathbf{Q}_{k+1} = \mathbf{H}_1^H \cdots \mathbf{H}_{r-1}^H\mathbf{H}_r^H$. Given $\mathbf{h}_i$ and $\mathbf{h}_i'$ for all $i = 1, 2, \ldots, r$, computing the left r columns of $\mathbf{H}_r^H$ requires nr flops, computing the left r columns of $\mathbf{H}_{r-1}^H\mathbf{H}_r^H$ requires an additional $2nr$ flops, and so on. To complete the accumulation of $\mathbf{Q}_{k+1}$, we need about $2nr^2$ flops.

Therefore, the Householder QR decomposition requires about $3nr^2$ flops and r square roots. (A flop defined in [1] is about half a flop defined here. The flop count shown in [1, pp. 225] is much higher than what we have shown here.)

Givens Transformation

The Givens transformation is essentially a 2×2 orthonormal matrix of the form $\mathbf{G} = \begin{bmatrix} c & s \\ -s^* & c^* \end{bmatrix}$ where c and s are such that

$$\mathbf{G}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sqrt{|a|^2 + |b|^2} \\ 0 \end{bmatrix},$$

i.e., $c = a^*/\sqrt{|a|^2 + |b|^2}$ and $s = b^*/\sqrt{|a|^2 + |b|^2}$. (The definitions of the Givens transformation for complex data may differ from each other by a diagonal matrix of complex numbers of unit amplitude.) If we wish to achieve

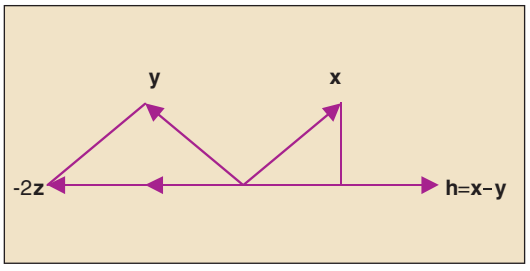
$$\mathbf{G}\begin{bmatrix} \vdots \\ x_i \\ \vdots \\ x_j \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \sqrt{|x_i|^2 + |x_j|^2} \\ \vdots \\ 0 \\ \vdots \end{bmatrix},$$

we should define

$$\mathbf{G} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & c & \mathbf{0} & s & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -s^* & \mathbf{0} & c^* & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$$

where c and s are located on the ith and jth rows and the ith and jth columns.

Using the Givens transformations, transforming $\hat{\mathbf{S}}_{k+1}$ to $\mathbf{R}_{k+1}$ should proceed as follows: zero the last element of the first column; continues upwards along the column until the second element becomes zero; then returns to the last element of the second column; continues until the third element of the second column is zero; and then returns to the last element of the next column; and repeats the process until all elements below the diagonal are zeroed. There are about nr zeros in $\mathbf{R}_{k+1}$, and each zero is achieved by a Givens transformation that



▲ *1. Illustration of the Householder transformation: transforming* **x** *into* **y** *of equal 2-norm, where* **z** *is the orthogonal projection vector of* **x** *onto the span of* **h**.

affects two rows, each of which has r elements. To complete the work for the first column of $\mathbf{R}_{k+1}$, we need about $4nr$ flops. To complete the work for the second column of $\mathbf{R}_{k+1}$, we need about $4n(r-1)$ flops. To complete the transformation from $\hat{\mathbf{S}}_{k+1}$ to $\mathbf{R}_{k+1}$ requires about $2nr^2$ flops.

Similar to the Householder method, constructing $\mathbf{S}_{k+1}$ simply needs to compute the left r columns of $\mathbf{Q}_{k+1} = \mathbf{G}_1^H \mathbf{G}_2^H \ldots \mathbf{G}_s^H$ where $s \approx nr$ and $\mathbf{G}_i$ for each $i = 1, 2, \ldots, s$ is a Givens transformation expressed in terms of two complex numbers. To obtain $\mathbf{S}_{k+1}$ from $\mathbf{G}_i$, we can do the following:

$$\begin{aligned}
&\mathbf{S}_{k+1,s} = \text{first_}r\text{_columns_of_}\mathbf{G}_s^H;\\
&\mathbf{S}_{k+1,i} = \mathbf{G}_i^H \mathbf{S}_{k+1,i+1} \qquad \text{for}\\
&\quad i = s-1, s-2, \ldots, 1;\\
&\mathbf{S}_{k+1} = \mathbf{S}_{k+1,1}
\end{aligned}$$

where each $\mathbf{G}_i^H \mathbf{S}_{k+1,i+1}$ requires about $4r$ flops. Hence, constructing $\mathbf{S}_{k+1}$ from $\mathbf{G}_i$, $i = 1, 2, \ldots, s$, requires about $4nr^2$ flops.

Therefore, the Givens QR decomposition requires about $6nr^2$ flops and nr square roots. An alternative method is to use the fast Givens transformation [1, p. 228], where square root is not required but additional operations are necessary to avoid numerical overflow (or underflow). The fast Givens transformation does not yield orthonormalization of $\mathbf{S}_{k+1}$ although ortho- gonalization is achieved.

Gram-Schmidt Transformation

The Gram-Schmidt method is perhaps the most classical method for QR decomposition. Given an $n \times r$ matrix $\mathbf{A}$, we now write the QR-decomposition $\mathbf{A} = \mathbf{QR}$ where $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2 \ldots \mathbf{a}_r]$ and $\mathbf{Q} = [\mathbf{q}_1\ \mathbf{q}_2 \ldots \mathbf{q}_r]$. To obtain an orthonormal $\mathbf{Q}$, we may use the following:

For $i = 1, 2, \ldots, r$, do

$$\begin{cases} \hat{\mathbf{q}}_i = \mathbf{a}_i - \sum_{k=1}^{i-1} \left(\mathbf{q}_k^H \mathbf{a}_i\right) \mathbf{q}_k \\ \mathbf{q}_i = \frac{\hat{\mathbf{q}}_i}{\|\hat{\mathbf{q}}_i\|} \end{cases}$$

End.

This method requires about nr^2 flops and r square roots, which is the most efficient among the methods discussed so far. More detailed implementations of the Gram-Schmidt method are given in [1, p. 231].

Alternatively, we can use the following method where the square roots are avoided. However, the resulting matrix $\mathbf{Q}$ is orthogonal but not orthonormal.

For $i = 1, 2, \ldots, r$, do

$$\mathbf{q}_i = \mathbf{a}_i - \sum_{k=1}^{i-1} \frac{\left(\mathbf{q}_k^H \mathbf{a}_i\right) \mathbf{q}_k}{\mathbf{q}_k^H \mathbf{q}_k}$$

End.

Method 2: Using Matrix Square-Root-Inverse

All QR-decomposition methods require square roots to obtain an orthonormal $\mathbf{S}_{k+1}$. To achieve an orthonormalization of $\mathbf{S}_{k+1}$, another choice of the function $f(\hat{\mathbf{S}}_{k+1}, \mathbf{S}_k)$ is

$$\mathbf{S}_{k+1} = \hat{\mathbf{S}}_{k+1} \left(\hat{\mathbf{S}}_{k+1}^H \hat{\mathbf{S}}_{k+1}\right)^{-1/2}$$

where the square-root-inverse is explicitly used to force the matrix $\mathbf{S}_{k+1}$ to be orthonormal. The computational complexity is about $(3/2)nr^2$ flops. (The factor $1/2$ is due to the symmetry of $\hat{\mathbf{S}}_{k+1}^H \hat{\mathbf{S}}_{k+1}$.) The square-root-inverse of the $r \times r$ matrix $\hat{\mathbf{S}}_{k+1}^H \hat{\mathbf{S}}_{k+1}$ costs $O\left(r^3\right)$ flops, which is negligible to $(3/2)nr^2$. We need to assume that $\hat{\mathbf{S}}_{k+1}^H \hat{\mathbf{S}}_{k+1}$ is invertible, which however is a weak condition in practice especially when $n \gg r$.

In Method 2, if $\mathbf{S}_k$ is multiplied from the right by a unitary matrix $\mathbf{Q}$, then the corresponding $\mathbf{S}_{k+1}$ is also multiplied from the right by $\mathbf{Q}$. But in Method 1, the corresponding $\mathbf{S}_{k+1}$ would be altered by a different unitary matrix, i.e., there is a "discontinuity" between $\mathbf{S}_k$ and $\mathbf{S}_{k+1}$. This property affects a critical approximation useful for an adaptive implementation of the power method [5]. We will briefly discuss such an adaptive implementation later.

If the eigendecomposition of a matrix is available, the square root of the matrix can be found by square-rooting the eigenvalues. Another algorithm for computing the square root of a positive-definite matrix $\mathbf{A}$, is the following (a complex version of [1, P11.2–4, p.571]):

$$\mathbf{Y}_{j+1} = \frac{1}{2}\left(\mathbf{Y}_j + \mathbf{A}\mathbf{Y}_j^{-H}\right)$$

where $\mathbf{Y}_j^{-H}$ is the conjugate transpose inverse of $\mathbf{Y}_j$.

To show that $\mathbf{Y}_j$ converges to $\mathbf{A}^{1/2}$ quadratically with the initial condition $\mathbf{Y}_0 = \mathbf{I}$, we first write the above equation as

$$\mathbf{Z}_{j+1} = \frac{1}{2}\left(\mathbf{Z}_j + \mathbf{Z}_j^{-H}\right)$$

where $\mathbf{Z}_j = \mathbf{A}^{-1/2}\mathbf{Y}_j$. Since $\mathbf{Y}_0 = \mathbf{I}$, and $\mathbf{A}$ and $\mathbf{A}^{1/2}$ are conjugate symmetric, then $\mathbf{Z}_0$ is conjugate symmetric. The above recursive equation further implies that $\mathbf{Z}_j$ is conjugate symmetric for all j. Furthermore, it follows that the eigenvectors of $\mathbf{Z}_j$ are invariant to j, and each eigenvalue of $\mathbf{Z}_j$ satisfies $\lambda_{j+1} = (\lambda_j + \lambda_j^{-1})/2$. Then, we can write

$$\lambda_{j+1} - 1 = \frac{1}{2\lambda_j}\left(\lambda_j - 1\right)^2.$$

From this equation, we know that each eigenvalue of the conjugate symmetric matrix $\mathbf{Z}_j$ converges to one, and hence $\mathbf{Z}_j$ converges to the identity matrix. Therefore, $\mathbf{Y}_j$ converges to $\mathbf{A}^{1/2}$. The convergence rate is obviously quadratic when $\left|\lambda_j - 1\right| < 1$. But when any of the eigenvalues is initially much larger than one, one iteration of the square root computation reduces

the eigenvalue λ_j by a factor 2. (If an initial eigenvalue is much smaller than one, the corresponding eigenvalue after one iteration becomes much larger than one.) In other words, the convergence of the square root algorithm is quadratic locally but linear otherwise.

Method 3: Using Matrix Inverse

In this method, no square root is used, and the function $f(\hat{\mathbf{S}}_{k+1}, \mathbf{S}_k)$ is defined by

$$\mathbf{S}_{k+1} = \hat{\mathbf{S}}_{k+1}\left(\mathbf{S}_k^H \hat{\mathbf{S}}_{k+1}\right)^{-1}.$$

The computational complexity at each iteration is $2nr^2$ flops. This normalization ensures that $\mathbf{S}_{k+1}$ has a bounded norm. But $\mathbf{S}_{k+1}$ is not forced to be orthonormal, or to converge to an orthonormal matrix. This method is a batch (as opposed to adaptive) form of the PAST algorithm [4], which was originally designed for adaptive subspace computation where the data matrix $\mathbf{X}$ is augmented by a new data vector at each iteration. The new data vector causes a series of rank-one updates in each iteration of the power method. Rank-one updates of matrix multiplication and matrix inverse are simple to implement, but rank-one update of matrix square root is not as simple (i.e., costing more). The use of inverse, instead of square root inverse, is a key reason that makes the adaptive form of Method 3 simpler than those of Method 2 [5]. The PAST algorithm has the lowest complexity in computation among all power-based adaptive algorithms with comparable performance in terms of subspace tracking error.

Method 4: Using Subspace Leakage

In this method, also known as NIC method [6], the function $f(\hat{\mathbf{S}}_{k+1}, \mathbf{S}_k)$ is expressed by

$$\mathbf{S}_{k+1} = (1-\eta)\mathbf{S}_k + \eta\hat{\mathbf{S}}_{k+1} \times \left(\mathbf{S}_k^H \hat{\mathbf{S}}_{k+1}\right)^{-1}$$

where $0 < \eta < 1$, and the computational complexity at each iteration is $2nr^2$. In this method, there is a leakage term $(1-\eta)\,\mathbf{S}_k$, i.e., a leakage from the old subspace matrix $\mathbf{S}_k$ to the new subspace matrix $\mathbf{S}_{k+1}$. A surprising property of the leakage is that the subspace matrix converges to an orthonormal matrix provided $0 < \eta < 1$. This result was proved for a very small η [6], [7]. But for any fixed value subject to $0 < \eta < 1$, a complete proof (or disproof) has remained open for several years [8]. When η is close to one, Method 4 has about the same performance for subspace tracking as Method 3, consumes about the same flops as Method 3, but has an asymptotically orthonormal property not shared by Method 3.

Method 5: Asymptotical Orthonormalization—A New Method

In this method, the function $f(\hat{\mathbf{S}}_{k+1}, \mathbf{S}_k)$ is defined by

$$\mathbf{S}_{k+1} = 2\hat{\mathbf{S}}_{k+1}\left(\mathbf{P}_{k+1}^2 + \mathbf{T}_{k+1}\right)^{-1} \times \mathbf{P}_{k+1} \qquad (1)$$

with

$$\mathbf{P}_{k+1} = \mathbf{S}_k^H \hat{\mathbf{S}}_{k+1}$$
$$\mathbf{T}_{k+1} = \hat{\mathbf{S}}_{k+1}^H \hat{\mathbf{S}}_{k+1}$$

where there is no square-root operation. It is easy to verify that the complexity of (1) is about $(5/2)nr^2$ flops at each iteration. Note that $\mathbf{P}_{k+1}$ uses nr^2 flops, $\mathbf{T}_{k+1}$ uses $(1/2)nr^2$ flops, $\mathbf{G}_{k+1} \triangleq 2(\mathbf{P}_{k+1}^2 + \mathbf{T}_{k+1})^{-1}$ $\mathbf{P}_{k+1}$ uses an additional $O(r^3)$ flops, and $\hat{\mathbf{S}}_{k+1}\mathbf{G}_{k+1}$ uses another nr^2 flops. Despite the seemingly complicated look of (1), Method 5 actually consumes less flops than the Householder QR method and the Givens QR method. Method 5 is no competitor to the Gram-Schmidt method in terms of flops. But Method 5 needs no square root operation.

Furthermore, for adaptive implementation of the power method, Method 5 yields a smoother transition from $\mathbf{S}_k$ to $\mathbf{S}_{k+1}$ than Method 1, which we briefly explain here. Assume that the data matrix $\mathbf{X}$ in the power method is replaced by a time-dependent data matrix $\mathbf{X}_k$ where k coincides with the iteration index of the power method. Also assume that the rank-r principal range of $\mathbf{X}_k$ remains constant. If $\mathbf{S}_k$ spans the same rank-r range, then one can verify that Method 5 leads to $\mathbf{S}_{k+1} = \mathbf{S}_k$ but Method 1 (i.e., any QR decomposition) in general makes $\mathbf{S}_{k+1}$ a "rotated" version of $\mathbf{S}_k$. This property of Method 5, as shared by Methods 2–4, allows the adaptive implementation of the power method to be further simplified with only a little performance penalty. For example, if we let the data matrix $\mathbf{X}_k$ be $\mathbf{X}_k = [\mathbf{X}_{k-1} \quad \mathbf{x}_k]$, then based on the power method, we can write the following:

$$\begin{aligned}\hat{\mathbf{S}}_{k+1} &= \mathbf{X}_k\mathbf{X}_k^H\mathbf{S}_k \\ &= \left(\mathbf{x}_k\mathbf{x}_k^H + \mathbf{X}_{k-1}\mathbf{X}_{k-1}^H\right)\mathbf{S}_k \\ &= \mathbf{x}_k\mathbf{x}_k^H\mathbf{S}_k + \mathbf{X}_{k-1}\mathbf{X}_{k-1}^H\mathbf{S}_k \\ &\approx \mathbf{x}_k\mathbf{x}_k^H\mathbf{S}_k + \mathbf{X}_{k-1}\mathbf{X}_{k-1}^H\mathbf{S}_{k-1} \\ &= \mathbf{x}_k\mathbf{x}_k^H\mathbf{S}_k + \hat{\mathbf{S}}_k\end{aligned}$$

from which we have a very efficient *subspace tracking algorithm*: $\hat{\mathbf{S}}_{k+1} = \mathbf{x}_k\mathbf{x}_k^H\mathbf{S}_k + \hat{\mathbf{S}}_k$ where the first term is an "innovation" and the second term a "propagation." A conventional forgetting factor may be further incorporated as follows: $\hat{\mathbf{S}}_{k+1} = (1-\alpha)\mathbf{x}_k\mathbf{x}_k^H\mathbf{S}_k + \alpha\hat{\mathbf{S}}_k$ where $0 < \alpha < 1$. Each iteration of the algorithm only needs $2nr$ flops plus the flops required for normalization. The accuracy of the subspace tracking algorithm depends on the accuracy of the approximation $\mathbf{X}_{k-1}$

Table 1. Comparison of different normalization methods. For the last column of the table, we assume that (a) $\mathbf{X}_k\mathbf{X}_k^H = \mathbf{U}_1\Pi_{1,k}\mathbf{U}_1^H + \mathbf{U}_2\Pi_{2,k}\mathbf{U}_2^H$ where $\mathbf{U}_1 \in \mathbf{C}^{n\times r}$ is an orthonormal matrix and orthogonal to $\mathbf{U}_2 \in \mathbf{C}^{n\times(n-r)}$; (b) $\Pi_{1,k}$ is nonsingular but not necessarily diagonal; and (c) $\hat{\mathbf{S}}_{k+1} = \mathbf{X}_k\mathbf{X}_k^H\mathbf{S}_k$ where $\mathbf{S}_k = \mathbf{U}_1\mathbf{V}_k$ and $\mathbf{V}_k$ is unitary. This assumption is an approximation of the case where the rank-r range of the subspace matrix $\mathbf{S}_k$ is close to the rank-r principal range of the data matrix $\mathbf{X}_k$.

Methods	Flops	Square roots	Orthonormalization	Condition for $\mathbf{S}_{k+1} = \mathbf{S}_K$
Method 1: (Any QR decomp.)	Givens QR: $6nr^2$ Householder QR: $3nr^2$ Gram-Schmidt QR: nr^2	Givens QR: nr Householder QR: r Gram-Schmidt QR: r	Yes	$\mathbf{V}_k$ is diagonal; or $\Pi_{1,k} = \pi_k\mathbf{I}$ for a scalar π_k.
Method 2	$1.5nr^2$	Square root of a $r \times r$ matrix	Yes	Any $\mathbf{V}_k$ and any $\Pi_{1,k}$.
Method 3	$2nr^2$	None	No	Any $\mathbf{V}_k$ and any $\Pi_{1,k}$.
Method 4	$2nr^2$	None	Yes, asymptotically. (linear rate)	Any $\mathbf{V}_k$ and any $\Pi_{1,k}$.
Method 5	$2.5nr^2$	None	Yes, asymptotically. (quadratic rate)	Any $\mathbf{V}_k$ and any $\Pi_{1,k}$.

$\mathbf{X}_{k-1}^H\mathbf{S}_k \approx \mathbf{X}_{k-1}\mathbf{X}_{k-1}^H\mathbf{S}_{k-1}$, which in turn depends on the choice of the normalization methods. The last column of Table 1 provides a good indication of how accurate the above approximation is for each of the normalization methods. Indeed, the above subspace tracking algorithm works well with Methods 2–5, but not as well with Method 1. (α should not be too small according to our experiment.)

Method 5 does not require square root but still has an orthonormal property. We show in the next section that $\mathbf{S}_{k+1}$ from Method 5 is asymptotically orthonormal, i.e., $\mathbf{S}_k^H\mathbf{S}_k \to \mathbf{I}$ for large k. The orthonormalization is achieved practically as soon as the principal subspace is found. Method 5 is a generalization of an algorithm shown in [9] where the $r = 1$ version of (1) was derived using a quasi-Newton search on the NIC cost function.

Unlike Method 4, Method 5 achieves asymptotical orthonormalization without any reduction of subspace tracking rate. As shown next, if the initialization $\mathbf{S}_0$ spans the desired principal subspace, Method 5 yields orthonormalization at a quadratic rate *globally*. Method 4 achieves orthonormalization at a linear rate [8].

Asymptotical Orthonormalization Property of Method 5

We now provide a proof of the asymptotical orthonormalization property of Method 5. The analytical (as opposed to "algorithmic) structure of Method 5 can be rewritten as

$$\mathbf{S}_{k+1} = \mathbf{C}\mathbf{S}_k\mathbf{G}_{k+1} \tag{2}$$

where $\mathbf{C} = \mathbf{X}\mathbf{X}^H \in C^{n\times n}$, and

$$\mathbf{G}_{k+1} = 2\left(\left(\mathbf{S}_k^H\mathbf{C}\mathbf{S}_k\right)\left(\mathbf{S}_k^H\mathbf{C}\mathbf{S}_k\right) + \mathbf{S}_k^H\mathbf{C}^2\mathbf{S}_k\right)^{-1}\left(\mathbf{S}_k^H\mathbf{C}\mathbf{S}_k\right) \in C^{r\times r}. \tag{3}$$

To simplify the structure of (2), we denote the eigendecomposition $\mathbf{C} = \mathbf{E}\Lambda\mathbf{E}^H$ where $\mathbf{E}$ is unitary and Λ is diagonal of descending eigenvalues, i.e., $\Lambda = \mathrm{diag}\left(\sigma_1^2\ \sigma_2^2\ \cdots\ \sigma_n^2\right)$. We also define the transformation of the subspace matrix: $\mathbf{V}_k = \mathbf{E}^H\mathbf{S}_k$, and write

$$\mathbf{V}_k = \begin{bmatrix}\mathbf{A}_k\\ \mathbf{B}_k\end{bmatrix}$$

where $\mathbf{A}_k \in C^{r\times r}$ and $\mathbf{B}_k \in C^{(n-r)\times r}$. Also let $\Lambda = \mathrm{diag}(\Lambda_A\Lambda_B)$ where Λ_A is $r \times r$ and Λ_B is $(n-r)\times(n-r)$. Then, it follows from (2) that

$$\mathbf{V}_{k+1} = \Lambda\mathbf{V}_k\mathbf{G}_{k+1}$$

and furthermore

$$\begin{cases}\mathbf{A}_{k+1} = \Lambda_A\mathbf{A}_k\mathbf{G}_{k+1}\\ \mathbf{B}_{k+1} = \Lambda_B\mathbf{B}_k\mathbf{G}_{k+1}.\end{cases} \tag{4}$$

We need to prove that $\mathbf{B}_k$ converges to zero and $\mathbf{A}_k$ converges to a unitary matrix under the following (weak) assumptions:

- 1) $\mathbf{A}_0$ is invertible
- 2) $\sigma_r^2 > \sigma_{r+1}^2$.

From (4), we have $\mathbf{B}_{k+1}\mathbf{A}_{k+1}^{-1} = \Lambda_B\mathbf{B}_k\mathbf{A}_k^{-1}\Lambda_A^{-1}$, and hence

$$\left(\mathbf{B}_{k+1}\mathbf{A}_{k+1}^{-1}\right)_{i,j} \le \left(\frac{\sigma_{r+1}}{\sigma_r}\right)^2 \times \left(\mathbf{B}_k\mathbf{A}_k^{-1}\right)_{i,j}. \tag{5}$$

Here, we have applied that $\mathbf{G}_{k+1}$ is invertible for any finite k, which is easy to verify. Since $\mathbf{A}_0$ is invertible,

then $\mathbf{G}_1$ is invertible according to (3), and then $\mathbf{A}_1$ is invertible according to (4), and then $\mathbf{G}_2$ is invertible according to (3), and so on. From (5), we know that $\mathbf{B}_k\mathbf{A}_k^{-1}$ can be arbitrarily small by choosing an arbitrarily large k.

We now write (2) as

$$\mathbf{S}_{k+1} = 2\mathbf{C}\mathbf{S}_k \left(\mathbf{S}_k^H\mathbf{C}\mathbf{S}_k\mathbf{S}_k^H\mathbf{C}\mathbf{S}_k + \mathbf{S}_k^H\mathbf{C}^2\mathbf{S}_k\right)^{-1}\mathbf{S}_k^H\mathbf{C}\mathbf{S}_k$$

or equivalently

$$\mathbf{V}_{k+1} = 2\Lambda\mathbf{V}_k \left(\mathbf{V}_k^H\Lambda\mathbf{V}_k\mathbf{V}_k^H\Lambda\mathbf{V}_k + \mathbf{V}_k^H\Lambda^2\mathbf{V}_k\right)^{-1}\mathbf{V}_k^H\Lambda\mathbf{V}_k.$$

Then, we have

$$\begin{aligned}\mathbf{A}_{k+1} =& 2\Lambda_A\mathbf{A}_k \\ &\times \left(\left(\mathbf{A}_k^H\Lambda_A\mathbf{A}_k + \mathbf{B}_k^H\Lambda_B\mathbf{B}_k\right)^2 \right. \\ &\left. + \left(\mathbf{A}_k^H\Lambda_A^2\mathbf{A}_k + \mathbf{B}_k^H\Lambda_B^2\mathbf{B}_k\right)\right)^{-1} \\ &\times \left(\left(\mathbf{A}_k^H\Lambda_A\mathbf{A}_k + \mathbf{B}_k^H\Lambda_B\mathbf{B}_k\right)\right)\end{aligned}$$

or equivalently,

$$\begin{aligned}\mathbf{A}_{k+1} =& 2\left((\mathbf{I}+\mathrm{E}_k)\mathbf{A}_k\mathbf{A}_k^H\right. \\ &\times \left.\left(\mathbf{I}+\mathrm{E}_k^H\right) + (\mathbf{I}+\Delta_k)\right)^{-1} \\ &\times (\mathbf{I}+\mathrm{E}_k)\mathbf{A}_k \qquad (6)\end{aligned}$$

with

$$\mathrm{E}_k = \Lambda_A^{-1}\left(\mathbf{B}_k\mathbf{A}_k^{-1}\right)^H\Lambda_B\mathbf{B}_k\mathbf{A}_k^{-1}$$
$$\Delta_k = \Lambda_A^{-1}\left(\mathbf{B}_k\mathbf{A}_k^{-1}\right)^H\Lambda_B^2\mathbf{B}_k\mathbf{A}_k^{-1}\Lambda_A^{-1}$$

According to (5), both E_k and Δ_k become arbitrarily small as k increases. Indeed, for large k, both E_k and Δ_k are in the order of $(\sigma_{r+1}/\sigma_r)^{4k}$. It follows from (6) that

$$\mathbf{A}_{k+1} = 2\left(\mathbf{A}_k\mathbf{A}_k^H + \mathbf{I}\right)^{-1}\mathbf{A}_k + \Theta_k$$

where Θ_k becomes arbitrarily smaller than the first term as k increases. This is because the perturbations E_k and Δ_k in (6) are all superimposed to the identity matrices. The exact expression of Θ_k can be found from (6) where a standard matrix inverse formula should be used. We omit this expression for simplicity of presentation. It then follows that

$$\begin{aligned}\mathbf{A}_{k+1}\mathbf{A}_{k+1}^H =& 4\left(\mathbf{A}_k\mathbf{A}_k^H + \mathbf{I}\right)^{-1} \\ &\times \mathbf{A}_k\mathbf{A}_k^H\left(\mathbf{A}_k\mathbf{A}_k^H + \mathbf{I}\right)^{-1} \\ &+ \Omega_k \qquad (7)\end{aligned}$$

where Ω_k is arbitrarily smaller than the first term as k increases. Denote an eigenvalue of $\mathbf{A}_k\mathbf{A}_k^H$ by λ_k. Then, from (7), we have

$$\lambda_{k+1} = \frac{4\lambda_k}{(\lambda_k+1)^2} + \varepsilon_k \qquad (8)$$

where $|\varepsilon_k|$ becomes arbitrarily smaller than the first term as k increases. Note that $0 < 4x/(x+1)^2 \le 1$ for $x > 0$, and the equality holds if and only if $x = 1$. Therefore, $4\lambda_k/(\lambda_k+1)^2$ in (8) is bounded between zero and one. Now, it follows from (8) that $|\varepsilon_k|$ must become arbitrarily small as k increases, and for $k \ge 0$

$$0 < \lambda_{k+1} - \varepsilon_k \le 1. \qquad (9)$$

This expression implies that the eigenvalues of $\mathbf{A}_k\mathbf{A}_k^H$ for $k > 0$ are bounded. Hence, (5) implies that $\mathbf{B}_k$ converges to zero.

Furthermore, from (8), we have

$$1 - \lambda_{k+1} = c_k(1-\lambda_k)^2 - \varepsilon_k \qquad (10)$$

where $0 < c_k = 1/(\lambda_k+1)^2 < 1$. Note that $-\varepsilon_{k-1} \le 1 - \lambda_k < 1$ where the first inequality follows from (9) and the second inequality follows from that $\mathbf{A}_k\mathbf{A}_k^H$ is positive-definite. Since $|\varepsilon_k| \to 0$, (10) implies that for large k, $1-\lambda_{k+1}$ is closer to zero than $1-\lambda_k$ up to the perturbation ε_k. Therefore, $\lambda_k \to 1$ and hence $\mathbf{A}_k$ converges to a unitary matrix.

The convergence rate of Method 5 is governed by (5). In fact, for large k, $|\varepsilon_k|$ is in the order of $(\sigma_{r+1}/\sigma_r)^{4k}$. If the initial $\mathbf{S}_0$ is such that $\mathbf{B}_0 = 0$, then $\mathbf{A}_k$ converges to a unitary matrix at a quadratic rate as governed by (10) with $\varepsilon_k = 0$. Since the subspace convergence rate is linear according to (5), the orthonormalization of the subspace matrix should be achieved practically as soon as the subspace matrix spans the desired principal subspace. Our numerical experiments have confirmed this observation.

Acknowledgment

The author thanks Gene H. Golub and anonymous reviewers for their helpful suggestions. S. Ouyang helped numerical verification of the property of Method 5.

Yingbo Hua is with the Department of Electrical Engineering at the University of California, Riverside.

References

[1] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: Johns Hopkins Univ. Press, 1996.

[2] G.W. Stewart, "Methods of simultaneous iteration for calculating eigenvectors of matrices," *Topics in Numerical Analysis*, J.H. Miller, Ed. New York: Academic, 1975, vol. 2, pp. 185–196.

[3] S. Ouyang and Y. Hua, "Bi-iterative least square method for subspace tracking," submitted for publication.

[4] B. Yang, "Projection approximation subspace tracking," *IEEE Trans. Signal Processing*, vol. 43, no. 1, pp. 95–107, 1995.

[5] Y. Hua, Y. Xiang, T. Chen, K. Abed-Meraim, and Y. Miao, "A new look at the power method for fast subspace tracking," *Digital Signal Process.*, vol. 9, no. 4, pp. 297–314, Oct. 1999.

[6] Y. Miao and Y. Hua, "Fast subspace tracking and neural network learning by a novel information criterion," *IEEE Trans. Signal Processing*, vol. 46, no. 7, pp. 1967–1979, 1998.

[7] W. Liu, W.-Y. Yan, V. Sreeram, and K.L. Tao, "Global convergence analysis for the NIC flow," *IEEE Trans. Signal Processing*, vol. 49, no. 10, pp. 2422–2430, Oct. 2001.

[8] Y. Hua and T. Chen, "On convergence of the NIC algorithm for subspace computation," *IEEE Trans. Signal Processing*, pp. 1112–1115, vol. 52, no. 4, Apr. 2004.

[9] S. Ouyang, P.C. Ching, and T. Lee, "Robust adaptive quasi-Newton algorithms for eigensubspace estimation," in *IEEE Proc. Vision, Image and Signal Processing*, Oct. 2003, vol. 150, no. 5, pp. 321–330.