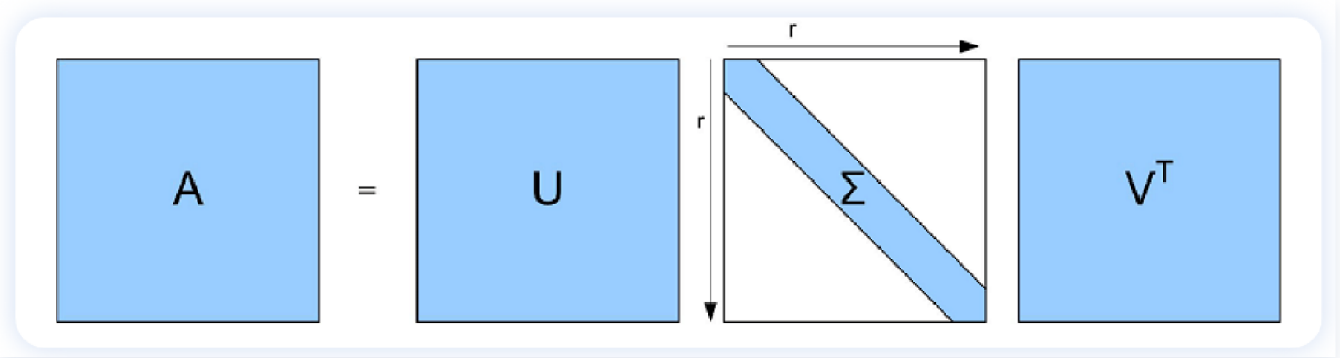
Singular value decomposition

The singular value decomposition of a matrix $A \in \mathbb{R}^{m \times n}$ is a factorization of the form $A = U\Sigma V^T$.



- $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are two orthogonal matrices.
- $\Sigma \in \mathbb{R}^{m \times n}$ is a pseudo-diagonal matrix with non-negative elements in decreasing order $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_p \geq 0$ with $p = \min(n, m)$

The elements σ_i of the matrix $\Sigma \in \mathbb{R}^{m \times n}$ are called **singular values** of A and correspond to $\sigma_i = \sqrt{\lambda_i}$, i.e., the square root of the eigenvalues of the matrix $A^T A$.

The columns u_i of the matrix U are the eigenvectors of AA^T and are called **left singular vectors**.

The columns v_i of the matrix V are the eigenvectors of $A^T A$ and are called **right singular vectors**.

Applications

This factorization reveals interesting properties of the matrix to which it is applied and allows us to:

- compute the pseudoinverse of a rectangular matrix (extends the concept of inverse to non-square matrices),
- estimate the condition number of the matrix,
- determine the rank of a matrix,
- approximate the initial matrix with a lower-rank matrix.

1. Rank

The number of nonzero singular values gives an indication of the rank of the matrix:

If $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r > \sigma_{r+1} = \ldots = \sigma_p = 0$,

that is, if r is the number of positive singular values, then $r = rank(A)$.

This allows us to rewrite A using its **spectral decomposition**:

> $$A = \sum_{i=1}^{r} u_i \sigma_i v_i^T.$$

2. Lower-rank approximation

Let $A = U\Sigma V^T$ with $A \in \mathbb{R}^{m \times n}$ and let $r = rank(A)$. Given a fixed $k < r$:

> $$A_k = u_1\sigma_1 v_1^T + u_2\sigma_2 v_2^T + \ldots u_k\sigma_k v_k^T.$$

A_k approximates the matrix A by truncating the spectral decomposition to a lower-rank k matrix.

The obtained approximation is the best approximation among all matrices B of rank k, i.e., the one with the minimum error:

$$\min_{B \in \beta} ||A - B||_2 = ||A - A_k||_2 = \sigma_{k+1}$$

$$\text{where } \beta = \{B \in \mathbb{R}^{m \times n} | rank(B) = k\}$$

Since σ_i are in decreasing order, as k increases, the approximation improves.

Lower-rank approximation is applied in **image compression**, as it reduces the number of necessary data for a good image reconstruction. This enables not only a reduction in file size, and thus storage space, but also a decrease in bandwidth usage in digital data transmission.

Singular value decomposition

To decompose a matrix into its SVD form, $A = U\Sigma V^T$, we should:

1. Determine the eigenvalues of the matrix $A^T A$.
2. Construct the matrix Σ as the square root of the eigenvalues.
3. Compute the matrix V^T as the eigenvectors of $A^T A$.
4. Compute the matrix U as the eigenvectors of AA^T.

Input matrix

A **digital image** is a representation of a physical image through a process of *sampling* and *quantization*. The image is sampled using a two-dimensional matrix of pixels (a single cell representing the minimal image unit), whose values are numerical representations defining its *intensity* or *grayscale level*.

At the end of this project, we will show how SVD can be used for image compression. In this case, the input matrix A to be decomposed will be a digital image.

1. Determine the eigenvalues of the matrix A^TA (1/2)

QR factorization can be used to compute the eigenvalues of a matrix.

Theorem: Let $A \in \mathbb{R}^{m \times n}$ be a matrix with all distinct eigenvalues in modulus $|\lambda_1| > |\lambda_2| > \ldots |\lambda_n|$, then the sequence:

$$k = 1, 2, \ldots \begin{cases} A_k = Q_k R_k \\ A_{k+1} = R_k Q_k \end{cases}$$

converges to a triangular matrix where $a_{ii}^{(k)} = \lambda_i$. If the matrix A is symmetric, it converges to a diagonal matrix.

This method follows a sequence of similar matrices, as A_{k+1} is orthogonally similar to A_k, which is orthogonally similar to A_{k-1} ... by transitivity, the final matrix has the same eigenvalues as A with the advantage that they are arranged along the diagonal.

Theorem: Two similar matrices A and $B = S^{-1}AS$ have the same eigenvalues.

Proof: $det(B - \lambda I) = det(B - \lambda S^{-1}S) = det(S^{-1}AS - \lambda S^{-1}S) = det(S^{-1}(A - \lambda I)S) =$

$= det(S^{-1}) * det(A - \lambda I) * det(S) = det(A - \lambda I)$

1. Determining the eigenvalues of the Matrix A^TA (2/2)

Lemma: Let $A_1 = Q_1 R_1$ and $A_2 = R_1 Q_1$. The matrices A_1 and A_2 are orthogonally similar and therefore have the same eigenvalues.

Proof: It is sufficient to note that $Q_1 A_2 Q_1^T = Q_1 R_1 Q_1 Q_1^T = Q_1 R_1 = A_1$.

To determine the SVD, we seek the eigenvalues of the matrix A^TA, which is ***symmetric*** by construction. Therefore, the iterative method converges to a diagonal matrix.

In the final iteration, we obtain the diagonalization $D = S^{-1}AS$, where $D = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_n \end{pmatrix}$

Writing $S = [\vec{v_1} \dots \vec{v_n}]$ and multiplying both sides of the equation by this matrix, we obtain $SD = AS$. On one hand, $SD = [\lambda_1 \vec{v_1}, \dots, \lambda_n \vec{v_n}]$ and on the other, $AS = A[\vec{v_1}, \dots, \vec{v_n}]$. Therefore, the equality suggests that $A\vec{v_i} = \lambda_i \vec{v_i}$, meaning that the column $\vec{v_i}$ is an eigenvector associated with the eigenvalue λ_i.

In the iterative method, the matrix S that diagonalizes our matrix is given by the application of the various Q_i, i.e, $D = Q_n^T \dots Q_2^T Q_1^T A Q_1 Q_2 \dots Q_n$. As a result, the eigenvectors corresponding to the eigenvalues, arranged along the diagonal, are precisely the column vectors of the product $S = \prod_i Q_i$.

2. Efficiency Considerations (1/2)

At each iteration step, the QR decomposition has a computational cost of $O(n^3)$. To reduce complexity, it is beneficial to transform the input matrix into an equivalent **Hessenberg form**. A matrix H is in *Hessenberg form* if all elements below the first subdiagonal are zero, i.e., $h_{ij} = 0$ for $i > j + 1$. Once a matrix is reduced to Hessenberg form at a cost of $O(n^3)$, performing QR decomposition at each step will require only $O(n^2)$, using *Givens rotations*.

$$H = \begin{bmatrix} X & X & X & X \\ X & X & X & X \\ 0 & X & X & X \\ 0 & 0 & X & X \end{bmatrix} \quad \textit{forma di Hessenberg}$$

The Hessenberg reduction is obtained by applying Householder transformations, excluding at each step the diagonal element (a_{kk}) and applying the transformation to the element at position (a_{k+1k}).

Consider $A^{(1)} = A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & A_{11} \end{pmatrix}$ and let $\overline{H_1}$ be the Householder transformation applied to the vector a_{21} so that $\overline{H_1} a_{21} = -\sigma_1 e \in \mathbb{R}^{(n-1)\times 1}$. We generate the matrix at the second step by pre-multiplying and post-multiplying with the Householder matrix H_1:

$$A^{(2)} = \begin{pmatrix} 1 & 0 \\ 0 & \overline{H_1} \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & A_{11} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \overline{H_1} \end{pmatrix}$$

This transformation eliminates all elements in the first column except for the first two.

By iterating this process $n - 2$ times, we obtain an upper Hessenberg matrix:

```
function [H, A] = hessemberg(A)
n=size(A,2);
H=eye(n);
for k=1:n-2
    % Apply the transformation excluding the first element of the k-th column,
    % starting from the index at position k+1
    sigma=sign(A(k+1,k))*norm(A(k+1:n,k));
```

```
        % v is the k-th column with the first element perturbed by sigma.
        v =[sigma+A(k+1,k);A(k+2:n,k)];
        beta =1/(sigma*(sigma+A(k+1,k)));

        % Transforms the lower part of the matrix A, that is, the transformation HA.
        for j=k:n
            tau = (v'*A(k+1:n,j))*beta;
            A(k+1:n,j)= A(k+1:n,j)-tau*v;
        end
        % Transforms the upper part of the matrix A, that is, the transformation AH.
        for j=1:n
            tau = (A(j,k+1:n)*v)*beta;
            A(j,k+1:n)= A(j,k+1:n)-tau*v';
            % We keep track of the transformation performed,
            % that is, the AH..Hn-3Hn-2, the matrix that post-multiplies A
            tau = (H(j,k+1:n)*v)*beta;
            H(j,k+1:n)= H(j,k+1:n)-tau*v';
        end
    end
```

2. Efficiency Considerations (2/2)

Once the initial matrix has been reduced to Hessenberg form, we apply the QR decomposition of an upper Hessenberg matrix $H = QR$ using **Givens rotations**.

An elementary Givens matrix is a plane rotation defined in matrix form as follows:

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}, \text{ where } c = cos(\theta) \text{ and } s = sin(\theta).$$

The *Givens* matrix is **orthogonal**, and therefore, it can be applied at each step of the iteration algorithm since it generates a matrix that is orthogonally similar to the original one. Moreover, its application **preserves the Hessenberg form**.

The transformation into an orthogonally similar matrix is achieved by applying $n-1$ Givens rotations $G_1, \ldots, G_{n-1}$, which transform H into an upper triangular form. Then, the same rotations are applied from the right to restore the Hessenberg form.

```
% returns the Givens matrix used to eliminate the element of A at position (j,i)
function [cos,sin] = givensRotations(A,i,j)
cos= abs(A(i,i)) / sqrt(A(i,i)^2+A(j,i)^2);
sin= -sign(A(j,i)/A(i,i)) * abs(A(j,i))/sqrt(A(i,i)^2+A(j,i)^2);
```

3. Accelerating Convergence

The convergence of the QR algorithm can be significantly improved by introducing *shift* values into the algorithm.

> $$k = 1, 2, \ldots \begin{cases} (A_k - s_k I) = Q_k R_k \\ A_{k+1} = R_k Q_k + s_k I \end{cases}$$

The optimal shift at the *k*-th step of the QR algorithm is given by using one of the eigenvalues of the matrix $H^{(k)}$. However, since the actual eigenvalues are unknown, various heuristics are employed. The most well-known are:

- **Rayleigh quotient shift**, which estimates the eigenvalue as the last diagonal element $s_k = h_{n,n}^{(k-1)}$
- **Wilkinson shift**, which coincides with the eigenvalue of the matrix $H^{(k)} = \begin{bmatrix} h_{n-1n-1}^k & h_{n-1n}^k \\ h_{nn-1}^k & h_{nn}^k \end{bmatrix}$

Eigenvalue Search Algorithm

Thus, the eigenvalue search procedure consists of:

1. Initially reducing the matrix to upper *Hessenberg* form.
2. Factorizing it at each step using *Givens* matrices, which preserve its structure.
3. Employing the *Wilkinson shift* to accelerate convergence.

In code form, it becomes:

```
function [Q,R] = qrfatt(A,toll)
[P,H]= hessemberg(A);
n=length(H);
% Q contains the eigenvectors.
Q=P;
```

```
G=zeros(n-1,2);
error=toll+1;
while error > toll
    % shift s=H(n,n);
    delta=(H(n-1,n-1)-H(n,n))/2;
    s=H(n,n)-sign(delta)*H(n,n-1)^2/(sign(delta)+sqrt(delta^2+H(n,n-1)^2));
    H=H-s*eye(n);
    for k=1:n-1
        % It takes the Givens rotations of the Gij matrix,
        % specific to eliminate the subdiagonal.
        [G(k,1),G(k,2)]= givensRotations(H,k,k+1);
        % apply the rotation to H, that is, perform the product Gij * H.
        H(k:k+1, k:n) = [G(k,1) -G(k,2); G(k,2) G(k,1)]*H(k:k+1, k:n);
    end

    for k=1:n-1
        % We post-multiply by the transposed Givens matrices
        % to obtain Hi+1 = G * A * G',
        % the Hessenberg matrix for the next iteration.
        H(1:k+1, k:k+1) = H(1:k+1, k:k+1)*[G(k,1) G(k,2); -G(k,2) G(k,1)];
        % Q keeps track of all the transformations performed:
        % Q = G12' * G23' * G34'...
        % Q=Q*G;
        Q(:, k:k+1) = Q(:, k:k+1)*[G(k,1) G(k,2); -G(k,2) G(k,1)];
    end
    H=H+s*eye(n);
    % as the method converges, the element h(n,n-1) will approach 0.
    error=abs(H(n,n-1));
end
R=H;
```

SVD Algorithm

qrfatt returns R containing the eigenvalues on the diagonal and Q the corresponding eigenvectors in the columns. We can proceed with the *SVD* decomposition:

```
function [U,S,V] = my_svd(A)
[m,n]=size(A);
[Q,R]=qrfatt(A*A',10^3);
% S is defined as the square root of the ordered eigenvalues.
S=sqrt(diag(R));
V=Q;
```

```
% The first column of V represents the eigenvector associated with the first
eigenvalue of S.
% When sorting the eigenvalues, we must maintain the correspondence.
if ~issorted(S,'descend')
    [S, indexs]=sort(S,'descend');
    V=V(:,indexs);
end
% We can derive U from the relation A = U*S*V'
% To do so, we need the inverse singular values.
S2=1./S(S~=0);
S2(n)=0;
U=A*V*diag(S2);
```

SVD on an RGB Image

Application of SVD compression to *'baboon'*:

```
A = imread('baboon.tiff');
[n,m,d]=size(A);
A=double(A);
R = A(:,:,1);
G = A(:,:,2);
B = A(:,:,3);
% I apply the SVD to each of the 3 channels.
[U_r,S_r,V_r] = my_svd(R);
[U_g,S_g,V_g] = my_svd(G);
[U_b,S_b,V_b] = my_svd(B);

K=[22, 55, 100, 320, 500];
for i=1:size(K,2)
    k=K(i);
    % I reconstruct each channel by considering the first k singular values.
    R_compr= U_r(:,1:k)*diag(S_r(1:k))*V_r(:,1:k)';
    G_compr= U_g(:,1:k)*diag(S_g(1:k))*V_g(:,1:k)';
    B_compr= U_b(:,1:k)*diag(S_b(1:k))*V_b(:,1:k)';
    % I reconstruct the entire image using the first k singular values.
    img=cat(3,R_compr(:,:,1),G_compr(:,:,1),B_compr(:,:,1));
    %Compression ratio
    ratio=(n*m)/(n*k+k+m*k);
    subplot(1,5,i),imshow(uint8(img)),title(['k:' num2str(k) '   Ratio:'
num2str(ratio)]);hold  on
end
```

Below are the reconstructions of the image using the first k singular values. As k increases, the reconstruction becomes sharper at the cost of higher memory usage.

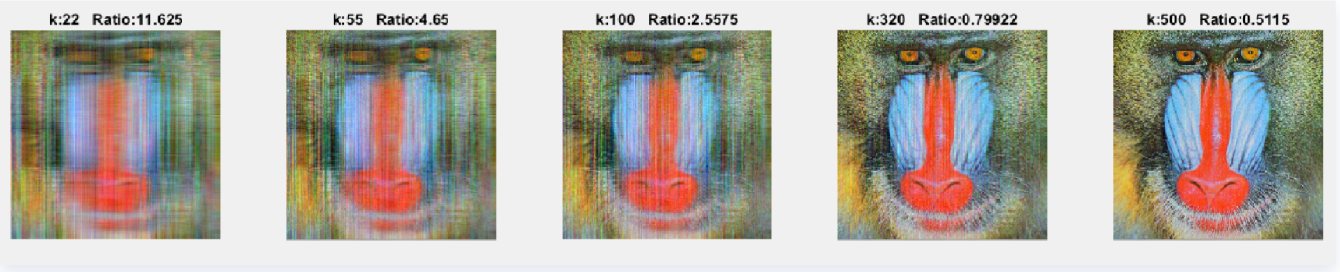
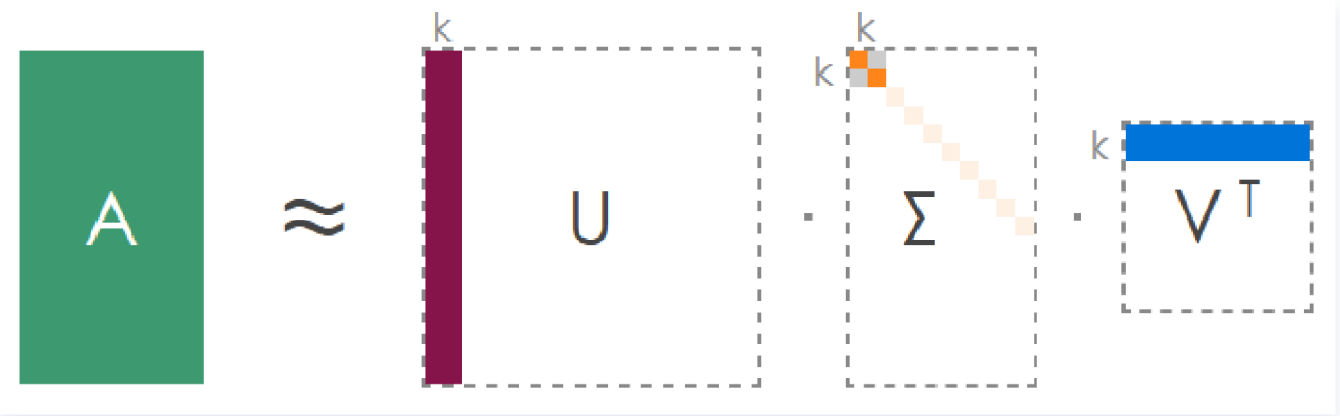


Image Compression

Rather than storing the $n \times m$ matrix, compression consists of stopping the spectral decomposition at the first k singular values, and keeping in memory the three matrices $U\Sigma V$ with total dimensions $n \times k + k + m \times k$.



SVD is also commonly used in statistics for principal component analysis and in numerical simulations to reduce the order of models.

References

- G. H. Golub and C. F. van Loan. Matrix Computations. The Johns Hopkins University Press, Baltimore, MD, 1996: 342-345
- Linear Algebra and Its Applications, 4th Edition, by Gilbert Strang
- Singular Value Decomposition: Applications to Image Processing Elizabeth A. Compton and Stacey L. Ernstberger
- Implementing the QR algorithm for efficiently computing matrix eigenvalues and eigenvectors, Gorka Erana Robles