Notes on Dynamic Mode Decomposition (DMD)

Vincent Moroney

I. Introduction

Dynamic Mode Decomposition (DMD) is a recently new (2008) algorithm for working with high dimensional data typically governed by parameterised systems and it's an analysis method highly related to other methods big data methods like SVD, PCA and ICA. This method was introduced in fluid mechanics in order to deal with highly dimensional fluid mechanics and to avoid direct numerical solutions to the Navier-Stokes equations. This set of notes is meant to be an overview of the DMD (and its relevant mathematical underpinnings) which contains a survey of the literature on the subject, and is intended to be more comprehensible to an interested undergraduate (like myself). I wrote these notes in order to put a lot of the ideas spread throughout the literature in one place while I was working on DMD based background extraction. Note that this introduction does not follow the historical development of DMD in the fluid mechanics community, and instead focuses on utilizing already existing data reduction methods – namely singular value decomposition (SVD).

Almost all of science and engineering focuses on analysis of **dynamical systems**. A dynamical system relates the change in a state $\vec{x} \in \mathbb{R}^n$ over time to the current state $\vec{x}$. A dynamical system can be represented by the following equation:

$$\boxed{\frac{d\vec{x}}{dt} = f(\vec{x}, t, u; \beta)} \tag{I.1}$$

Here f is a vector valued function which details the evolution of a state, $\vec{x}$, over time t. The variable u represents some control over the dynamics such as feedback, natural responses of the state to previous states in time or forcing parameters. Finally β indicates other parameters that the dynamics of the system are dependent on. Of course since this method was developed initially for use in fluid mechanics, the dynamics (f) of the state evolution in time is not necessarily linear, and in fact DMD mostly finds application in the cases where the dynamics of the system are non-linear or unknown.

For most of the history of engineering and science, studies of (1) were focused on parameterised based analysis: Newton's second law relates the change in the momentum state in time to the current acceleration state of the system; Maxwell's equations relate changes in electric and magnetic fields (spatio-temporally) to things like the current states charge distribution, magnetic field changes and current sources; Schrodingers equation relates the time evolution of the wave function to the curvature of the system and the current wave function of the system. Nearly any system studied in engineering or physics can be understood as a dynamical system of the form (1). However, this parametric approach requires the knowledge of, or the ability to derive, the dynamics of a system before hand. Take the example of finding the charge density on a plate: For hundreds of years and even now the approach is to solve the integral equation corresponding to Laplace's equation over the plate with some initial potential conditions. This means it was required to (1) know that Laplace's equation describes in some way the charge distribution over the plate, (2) we were able to directly measure the potential on the boundary of the problem, and (3) the problem was feasibly analytically or numerically solveable. Take as another example finding the allowable modes in a certain geometry of transmission line, the approach would be to find the eigenmodes of the geometry which satisfy Maxwell's equations numerically or analytically. The goal of dynamic mode decomposition is to use already existing big data analysis methods to discover things like the eigenmodes of a system, or its dynamics over time which are deeply embedded into low rank structures **without any reference to the underlying dynamics of the system** and hence without ever having to deal with sometimes impossible to solve coupled partial differential equations as in the case of fluid mechanics. Indeed, one of the main strengths of DMD is its ability to uncover the dynamics of a system without ever knowing the mathematical formulation of the dynamics in the first place. The method also can be used when measurements of the system are imprecise, noisy or if it is otherwise impossible to directly measure the state of the system.

Another note of merit for the DMD is that, as in most big data methods, DMD addresses the tension that exists between how we represent and store data and the actual dynamics embedded in said data. Many engineers and scientists likely have felt this tension before: In images or audio we store the samples in arrays with sizes approaching the millions, only to discover that the dynamics of the system are sparse in the frequency domain; in fluid mechanics fluid flows often exhibit low dimensional behaviour despite the mechanics themselves being governed by infinite dimensional partial equations; nearly any "natural" system or signal can be approximated with considerably less data in some corresponding transform basis. Since the DMD often is implemented with sparsity oriented algorithms, or with dimensionality reduction methods, it is often possible to extract the dynamics of a system which otherwise is encoded in a million by million matrix with a 500 by 500 matrix.

II. Dynamic Mode Decomposition Through the Lense of Video Processing

Okay so what is DMD? The approach of this introduction is going to focus on the example of processing video, though there are probably many other suitable applications to introduce this topic. Imagine we have a video file with m greyscale frames that we wish to process. Each frame is an image – a **snapshot** – of the objects being recorded. We can call each of these snapshots a state of the video, each separated by a constant time Δt. The state is constructed by flattening the greyscale image array into a one dimensional array as shown in the image below.

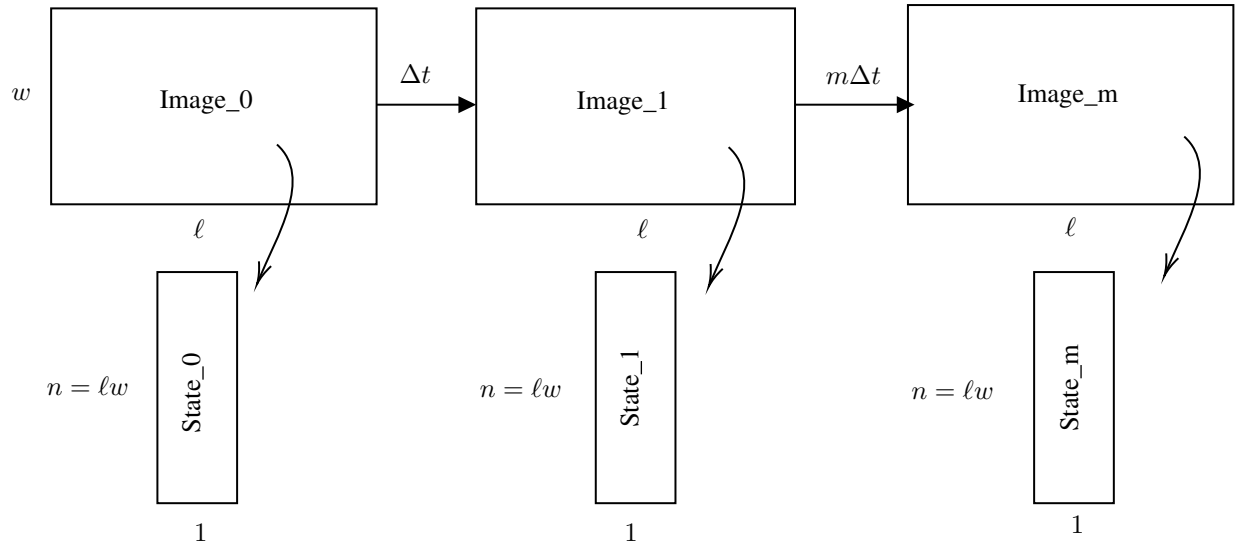

Fig. 1. We take m images of size $\ell \times w$ and compress them into $1 \times n$ column vectors where $n = \ell \times w$ and each of these column vectors describe a state of the video. Note this could easily be a fluid velocity field, the potential on a plate over time, etc.

We denote each of these states $\vec{x}$ and we assume that the relationship between one state and the next is governed by (1):

$$\frac{d\vec{x}}{dt} = f(\vec{x}, t, u; \beta)$$

We group all of the states as follows: We construct one matrix, $\mathbf{X} \in \mathbb{R}^{n \times m}$ which describes the states $0 - (m-1)$ and another matrix $\mathbf{Y} \in \mathbb{R}^{n \times m}$ which describes the states $1 - m$, i.e each jth column of $\mathbf{Y}$ corresponds to the frame after the jth column in $\mathbf{X}$. Explicitly if $\vec{x}_j$ corresponds to the jth state (frame) of the video then,

$$\mathbf{X} = \begin{bmatrix} | & | & \vdots & | \\ \vec{x}_0 & \vec{x}_1 & \vdots & \vec{x}_{m-1} \\ | & | & \vdots & | \end{bmatrix} \tag{II.1}$$

$$\mathbf{Y} = \begin{bmatrix} | & | & \vdots & | \\ \vec{x}_1 & \vec{x}_1 & \vdots & \vec{x}_m \\ | & | & \vdots & | \end{bmatrix} \tag{II.2}$$

Meaning we can view $\mathbf{X}$ and $\mathbf{Y}$ as having some dynamical causal relationship. However, the dynamics of how the pixels in one frame change into the pixels in the next frame are completely unknown. Attempting to derive these dynamics, if possible, wouldn't be useful either since they would likely be unique for any given video.

DMD helps us by asking the following question: Can we best fit a linear system, $\mathbf{A}$, which (in a least squares sense) takes the columns from $\mathbf{X}$ into the corresponding columns in $\mathbf{Y}$? Mathematically we wish to approximate (I) via:

$$\frac{d\vec{x}}{dt} = \mathbf{A}\vec{x} \tag{II.3}$$

which has a simple solution for the evolution of the dynamics in time:

$$\vec{x}(t) = \Phi \exp(\Omega t) \Phi^{\dagger} \vec{x}(0) \tag{II.4}$$

where Φ and Ω will be explained soon and $\vec{x}(0)$ is the zeroth frame[1]. Note that (II.3) and (II.4) together compose an incredible result hinted at in the introduction: A **parameterless** description of the dynamics at hand. In this case the dynamics at hand are how the pixels change between frames, but in the context of fluid mechanics the eigenvalues of $\mathbf{A}$ would encode the fundamental modes composing the fluid flow being analysed – without ever looking at Navier-Stokes. The DMD returns eigenmodes and eigenvalues of the dynamical system at hand, and can even reconstruct and predict future states as evidenced by (II.4).

We compose the following matrix problem equivalent to (II.3):

$$\mathbf{Y} = \mathbf{A}\mathbf{X} \tag{II.5}$$

where the error between the true $\mathbf{Y}$ and the computed $\mathbf{Y}$ is minimal in the least squares sense. Since the Moore-Penrose psuedoinverse has the property of least-squares optimization for the system (II.5), in theory the solution to this problem is as easy as:

$$\mathbf{A} = \mathbf{Y}\mathbf{X}^{\dagger} \tag{II.6}$$

However, consider that $\mathbf{X}$ and $\mathbf{Y}$ have size $n \times m$ indicating that $\mathbf{A}$ has size $n \times n$. Even if the frames have small size, say 320×240, and the video is composed of only 60 seconds of footage at 30fps, then we have a matrix $\mathbf{A}$ of size 76800×76800 and $\mathbf{X}$, $\mathbf{Y}$ have size 76800×1800. The eigen-decomposition of $\mathbf{A}$ which would encode the dynamics of (II.3) would likely be impossible to compute directly on a personal computer[2] . If you have a few thousand dollars laying around for access to cloud computing, then go right ahead with computing the eigen-decomposition of (II.6), however for anyone else wishing to do this with a simple IDE and personal computer, we need some form of dimensionality reduction. To get a scale of how large $\mathbf{A}$ can get, one of the largest fluid flows computed using a similar method took millions of CPU hours to resolve.

The need for dimensionality reduction is exactly how singular value decomposition (see appendix for SVD) found its way into the algorithm for computing the DMD. We can write the matrix $\mathbf{X}$ via its singular value decomposition:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{*} \tag{II.7}$$

Note that here $*$ indicates the conjugate transpose operator (often it reduces to the normal transpose operator). Then we can project $\mathbf{A}$ onto the subspace $\mathbf{U}$:

$$\tilde{\mathbf{A}} = \mathbf{U}^{*}\mathbf{A}\mathbf{U} \tag{II.8}$$

to form the low rank approximation for $\mathbf{A}$. Now note that $\mathbf{A} = \mathbf{Y}\mathbf{X}^{\dagger} = \mathbf{Y}\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^{\dagger}$, and further note that the inverse on $\mathbf{\Sigma}$ reduces to the normal inverse since $\mathbf{\Sigma}$ is always square. Then we can write

$$\boxed{\tilde{\mathbf{A}} = \mathbf{U}^{*}\mathbf{Y}\mathbf{V}\mathbf{\Sigma}^{-1}} \tag{II.9}$$

Now a VERY useful mathematical theorem that I will not attempt to prove here states the following: **The eigenvalues and eigenvectors of (II.9) are identical to the eigenvalues of (II.6).** This theorem further justifies the decomposition of $\mathbf{X}$ into its SVD and the projection of $\mathbf{A}$ onto $\mathbf{U}$ as we can take a rank r truncated SVD to greatly reduce the dimensionality of the problem at hand, and the r eigenvectors resulting from $\tilde{\mathbf{A}}$ will be very similar to the corresponding eigenvectors in $\mathbf{A}$.

Okay pausing for a moment to recap: The eigenvalues and eigenvectors of $\tilde{\mathbf{A}}$ describe the fundamental dynamics of the system (II.5), and the size of $\tilde{\mathbf{A}}$ as compared to $\mathbf{A}$ is greatly reduced allowing for realistic computation of said eigenvalues.

The eigenvectors and eigenvalues of $\tilde{\mathbf{A}}$ are extracted from the following equation:

$$\tilde{\mathbf{A}}\vec{w}_i = \lambda_i \vec{w}_i \tag{II.10}$$

The eigen-decomposition of $\mathbf{A}$ can be reconstructed as the following generalized eigen-decomposition:

$$\mathbf{A}\mathbf{\Phi} = \mathbf{\Phi}\mathbf{\Lambda} \text{ where } \boxed{\mathbf{\Phi} = \mathbf{Y}\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{W}} \tag{II.11}$$

where $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues and $\mathbf{W}$ is the matrix whose columns are eigenvectors of $\tilde{\mathbf{A}}$. $\mathbf{\Phi}$ are called the **DMD modes** and the columns of $\mathbf{\Phi}$ are the eigenvectors of $\mathbf{A}$.

At this point we can start to understand the equation (II.4), however before we get right into the continuous $\vec{x}(t)$ solution, let's consider a discrete time solution. The first frame is related to the zeroth frame via the following equation:

$$\vec{x}_1 = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{\dagger}\vec{x}_0 \tag{II.12}$$

[1]Note that $\mathbf{A}^{\dagger}$ refers to the Moore-Penrose psuedoinverse of $\mathbf{A}$ whereby $\mathbf{A}\mathbf{A}^{\dagger} = \mathbf{I}$.

[2]Note: Directly computing (II.6) is known as the **exact DMD** and was the method first introduced into the literature. This method is exactly how I BSOD'd my computer attempting background extraction for highway footage.

Then the second frame can be obtained by applying the same linear transformation again, but since we have written $\mathbf{A}$ as an eigenvalue decomposition, the linear transformation applying again only increases the degree on the diagonal eigenvalue matrix:

$$\vec{x}_2 = \mathbf{\Phi\Lambda\Phi}^\dagger \vec{x}_1 = \mathbf{\Phi\Lambda\Phi}^\dagger(\mathbf{\Phi\Lambda\Phi}^\dagger \vec{x}_0) = \mathbf{\Phi\Lambda}^2\mathbf{\Phi}^\dagger \vec{x}_0 \tag{II.13}$$

Repeating this process p times gives us the pth frame in the video:

$$\vec{x}_p = \mathbf{\Phi\Lambda}^p\mathbf{\Phi}^\dagger \vec{x}_0 \tag{II.14}$$

However, each frame is exactly Δt seconds away from any other frame. Therefore we have $t = p\Delta t$ and we can write:

$$\vec{x}(t) = \mathbf{\Phi\Lambda}^{t/\Delta t}\mathbf{\Phi}^\dagger \vec{x}(0) \tag{II.15}$$

This result is equivalent to (II.3). We define the **continuous eigenvalues** of the DMD modes as follows:

$$\boxed{\mathbf{\Omega} = \frac{\ln(\mathbf{\Lambda})}{\Delta t}} \tag{II.16}$$

Then we find

$$\mathbf{\Lambda} = \exp(\mathbf{\Omega}\Delta t) \tag{II.17}$$

Upon substitution of (II.17) into (II.15) we obtain

$$\vec{x}(t) = \mathbf{\Phi} \exp(\mathbf{\Omega}\Delta t)^{t/\Delta t}\, \mathbf{\Phi}^\dagger \vec{x}(0) \tag{II.18}$$

Which of course reduces via the rules of exponents to yield:

$$\boxed{\vec{x}(t) = \mathbf{\Phi} \exp\left(\mathbf{\Omega}\vec{t}\,\right) \mathbf{\Phi}^\dagger \vec{x}(0)} \tag{II.19}$$

It is common in the literature to denote the product $\mathbf{\Phi}^\dagger \vec{x}(0)$ as the quantity $\vec{b}$ to obtain

$$\vec{x}(t) = \mathbf{\Phi} \exp\left(\mathbf{\Omega}\vec{t}\,\right) \vec{b} \tag{II.20}$$

and sometimes $\vec{b}$ is referred to as the amplitude vector or the initial condition. This approach works for basically any dynamical system, not just this video processing example.

Remarks:

(1) Generally, evaluating (II.15) or (II.19) at some time t only gives an approximation of the state at said time. In the case of this image example if t was chosen to be an integer multiple of Δt then the state returned would be exact. Further, when low rank approximations to the data are taken, often modes will appear with slightly positive real values. Overtime the quality of reconstruction will degrade as these modes increase.

(2) The eigenvalues given by (II.16) form what's called a **generalized Laplace analysis** of the system at hand. These eigenvalues have the exact same interpretation as in Laplace analysis. If ω_j is an eigenvalue in $\mathbf{\Omega}$, and if $\text{Re}(\omega_j) > 0$, then the mode corresponding to this eigenvalue is said to be **unstable**. If $\text{Re}(\omega_j) = 0$ then the mode oscillates forever. If $\text{Re}(\omega_j) < 0$ then the mode exponentially decays to zero. This is because of the fundamentally exponential nature of the solution (II.19) and more deeply because DMD is related to **Koopman analysis**.

(3) In general DMD modes are not orthogonal.

(4) In the case of image processing in this note set, the state $\vec{x}$ can be directly observed, i.e each frame can directly be read from an mp4 video. If $\vec{y}$ is our measurement of the states, then in this case $\vec{y} = \vec{x}$ for each time step. However, sometimes the state can only be indirectly observed and $\vec{y} = g(\vec{x})$. DMD still works in this case.

Since we know that the eigenvalues (II.16) are interpreted analogously to traditional engineering spectral analysis, we can now easily arrive at background extraction. Remember, the dynamical system we've constructed is the relationship between the pixel values and time (frames). If over the course of the video there was, on average, regions on each frame which did not change in value over the duration of the video, then we can expect eigenvalues from the DMD satisfying $|\omega| \approx 0$. The approach then (at least for simple videos) is straight forward. Identify the background frequency of the video and propagate the background through each frame, then subtract the DMD reconstruction of the background from the original signal. The result should only be the moving objects over the course of the video. Of course this won't exactly be the best approach if the camera is shaky, if there's intense weather in the video or if the background itself is moving. However for simple footage this approach should be good enough.

III. Source Material and Further Reading

While I didn't do my due diligence and cite things properly in this document, the following sources were used to compile this document.

1) Textbook: Data Driven Science and Engineering: http://databookuw.com/
2) Compressed Dynamic Mode Decomposition for Background Modeling: https://link.springer.com/article/10.1007/s11554-016-0655-2
3) A masters thesis on DMD: https://dc.uwm.edu/cgi/viewcontent.cgi?article=2884&context=etd
4) An introductory lecture on DMD from Professor Nathan Kutz: https://www.youtube.com/watch?v=bYfGVQ1Sg98
5) An overview of DMD from Professor Steve Brunton: https://www.youtube.com/watch?v=sQvrK8AGCAo
6) A great blog post on DMD: http://www.pyrunner.com/weblog/2016/07/25/dmd-python/
7) A great introduction to dynamical systems: https://www.youtube.com/watch?v=-FvrON0OmYc
8) An overview of SVD from Professor Steve Brunton: https://www.youtube.com/watch?v=nbBvuuNVfco
9) Obligatory Wikipedia overview: https://en.wikipedia.org/wiki/Dynamic_mode_decomposition

Without these resources it would have taken me infinitely more time to implement this method.

APPENDIX

Appendix – Overview of Singular Value Decomposition (SVD)

The singular value decomposition (SVD) is a generalization of the eigenvalue decomposition. However, before getting into that let's talk about what the SVD even is. It finds use in pretty much any machine learning application or "big data" analysis method like DMD, PCA and ICA for rank reduction of big data. Suppose we have a matrix (typically large and under/over determined) $\mathbf{A} \in \mathbb{C}^{n \times m}$:

$$\mathbf{A} = \begin{bmatrix} | & | & \vdots & | \\ \vec{a}_0 & \vec{a}_1 & \vdots & \vec{a}_m \\ | & | & \vdots & | \end{bmatrix} \tag{A.1}$$

Each jth column of $\mathbf{A}$, $\vec{a}_j$, could represent (as in the case of DMD) a measurement of the state of a dynamical system at the jth time step. These columns, depending on which STEM field you're in, are typically called **snapshots** due to the common use of SVD in dynamical systems. Anyways, the SVD is a **unique** matrix decomposition that **must exist** for any $\mathbf{A}$ and it is typically written in the form:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}V^* \tag{A.2}$$

where $\mathbf{U} \in \mathbb{C}^{n \times n}$ and $\mathbf{V} \in \mathbb{C}^{m \times m}$ have the property of being **unitary**. If $\mathbf{U}$ is a unitary matrix then it satisfies $\mathbf{U}^*\mathbf{U} = \mathbf{U}\mathbf{U}^* = \mathbf{I}$ where $\mathbf{I}$ is the identity matrix. The columns of these SVD matrices $\mathbf{U}$ and $\mathbf{V}$ are mutually orthogonal. The matrix $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$ has real, positive values on the diagonal and zeros everywhere else, i.e if σ_{ij} is an entry in $\mathbf{\Sigma}$ then

$$\sigma_{ij} = 0 \iff i \neq j \text{ and } \sigma_{ii} > 0 \tag{A.3}$$

An important property of the $\mathbf{\Sigma}$ matrix is that when $n \geq m$ (which is almost always the case in engineering contexts), then the matrix $\mathbf{\Sigma}$ has **at most** m non zero elements on the diagonal. This allows for the construction of the **economy SVD** whereby $\mathbf{A}$ is written as:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \begin{bmatrix} \hat{\mathbf{U}} & \hat{\mathbf{U}}^{\perp} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ 0 \end{bmatrix} \mathbf{V}^* = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}V^* \tag{A.4}$$

The textbook Data Driven Science and Engineering (linked in the last section above) has a great illustration of the difference between the SVD and the economy SVD:

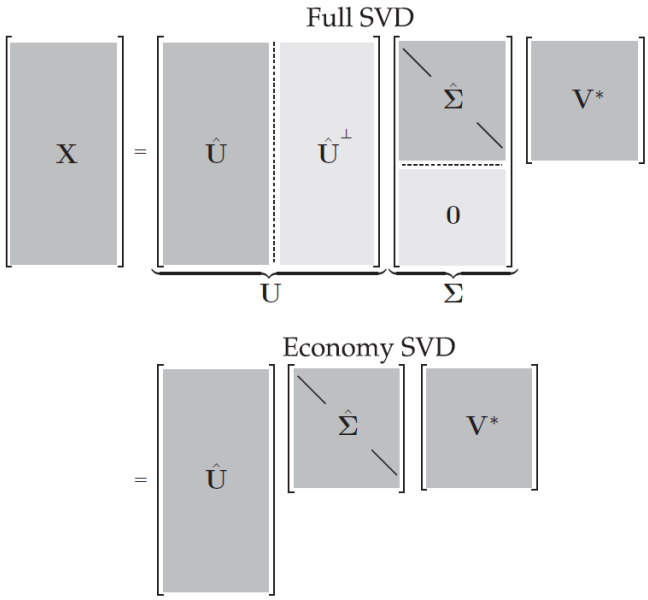

Fig. 2. The difference between the SVD and the economy SVD. It's required that $n \geq m$.

The matrix $\hat{\mathbf{U}}$ contains columns called the **left singular vectors** of $\mathbf{A}$ and the columns of $\mathbf{V}$ are called the **right singular vectors** of $\mathbf{A}$. The diagonal values of $\mathbf{\Sigma}$ are called the **singular values** of $\mathbf{A}$.