15 Linear Algebra and Singular Value Decomposition

Linear algebra plays a central role in almost every application area of mathematics in the physical, engineering and biological sciences. It is perhaps the most important theoretical framework to be familiar with as a student of mathematics. Thus it is no surprise that it also plays a key role in data analysis and computation. In what follows, emphasis will be placed squarely on the *singular value decomposition* (SVD). This concept is often untouched in undergraduate courses in linear algebra, yet it forms one of the most powerful techniques for analyzing a myriad of application areas.

15.1 Basics of the Singular Value Decomposition (SVD)

In even the earliest experience of linear algebra, the concept of a matrix transforming a vector via multiplication was defined. For instance, the vector $\mathbf{x}$ when multiplied by a matrix $\mathbf{A}$ produces a new vector $\mathbf{y}$ that is now aligned, generically, in a new direction with a new length. To be more specific, the following example illustrates a particular transformation:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 2 & 1 \\ -1 & 1 \end{bmatrix} \quad \rightarrow \quad \mathbf{y} = \mathbf{A}\mathbf{x} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}. \tag{15.1.1}$$

Figure 15.1(a) shows the vector $\mathbf{x}$ and its transformed version, $\mathbf{y}$, after application of the matrix $\mathbf{A}$. Thus generically, matrix multiplication will rotate and stretch (compress) a given vector as prescribed by the matrix $\mathbf{A}$ (see Fig. 15.1(a)).

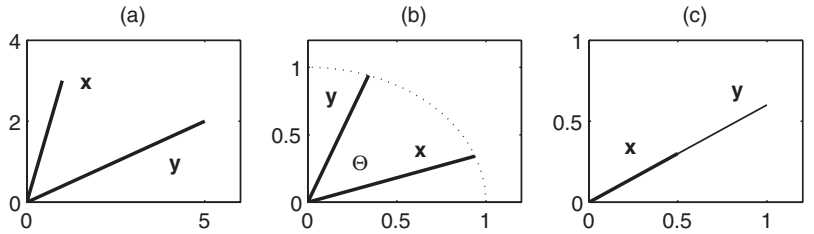

Figure 15.1: Transformation of a vector $\mathbf{x}$ under the action of multiplication by the matrix $\mathbf{A}$, i.e. $\mathbf{y} = \mathbf{A}\mathbf{x}$. (a) Generic rotation and stretching of the vector as given by Eq. (15.1.1). (b) Rotation by 50° of a unit vector by the rotation matrix (15.1.2). (c) Stretching of a vector to double its length using (15.1.3) with $\alpha = 2$.

The rotation and stretching of a transformation can be precisely controlled by proper construction of the matrix $\mathbf{A}$. In particular, it is well known that in a two-dimensional space, the rotation matrix

$$\mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{15.1.2}$$

takes a given vector $\mathbf{x}$ and rotates it by an angle θ to produce the vector $\mathbf{y}$. The transformation produced by $\mathbf{A}$ is known as a *unitary transformation* since the matrix inverse is $\mathbf{A}^{-1} = \bar{\mathbf{A}}^T$ where the bar denotes complex conjugation [30]. Thus rotation can be directly specified without the vector being scaled. To scale the vector in length, the matrix

$$\mathbf{A} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix} \tag{15.1.3}$$

can be applied to the vector $\mathbf{x}$. This multiplies the length of the vector $\mathbf{x}$ by α. If $\alpha = 2$ (0.5), then the vector is twice (half) its original length. The combination of the above two matrices gives arbitrary control of rotation and scaling in a two-dimentional vector space. Figure 15.1 demonstrates some of the various operations associated with the above matrix transformations.

A *singular value decomposition* (SVD) is a factorization of a matrix into a number of constitutive components all of which have a specific meaning in applications. The SVD, much as illustrated in the preceding paragraph, is essentially a transformation that stretches/compresses and rotates a given set of vectors. In particular, the following geometric principle will guide our forthcoming discussion: the image of a unit sphere under any $m \times n$ matrix is a hyper-ellipse. A hyper-ellipse in $\mathbb{R}^m$ is defined by the surface obtained upon stretching a unit sphere in $\mathbb{R}^m$ by some factors $\sigma_1, \sigma_2, \cdots, \sigma_m$ in the orthogonal directions $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_m \in \mathbb{R}^m$. The stretchings σ_i can possibly be zero. For convenience, consider the $\mathbf{u}_j$ to be unit vectors so that $\|\mathbf{u}_j\|_2 = 1$. The quantities $\sigma_j \mathbf{u}_j$ are then the *principal semi-axes* of the hyper-ellipse with the length σ_j. Figure 15.2 demonstrates a particular hyper-ellipse created under the matrix transformation $\mathbf{A}$ in $\mathbb{R}^2$.

A few things are worth noting at this point. First, if $\mathbf{A}$ has rank r, exactly r of the lengths σ_j will be nonzero. And if the matrix $\mathbf{A}$ is an $m \times n$ matrix where $m > n$, at most n of the σ_j will be nonzero. Consider for the moment a full rank matrix $\mathbf{A}$. Then the n singular values of $\mathbf{A}$ are the

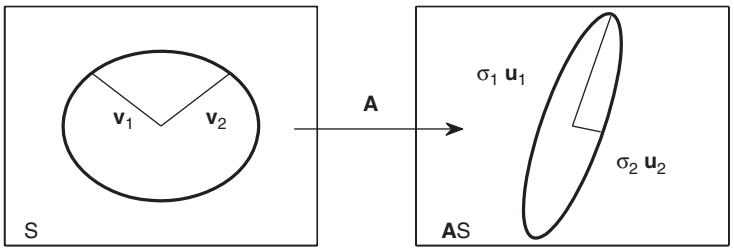

Figure 15.2: Image of a unit sphere S transformed into a hyper-ellipse $\mathbf{A}S$ in $\mathbb{R}^2$. The values of σ_1 and σ_2 are the singular values of the matrix $\mathbf{A}$ and represent the lengths of the semi-axes of the ellipse.

lengths of the principal semi-axes $\mathbf{A}S$ as shown in Fig. 15.2. Convention assumes that the singular values are ordered with the largest first and then in descending order: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$.

On a more formal level, the transformation from the unit sphere to the hyper-ellipse can be more succinctly stated as follows:

$$\mathbf{A}\mathbf{v}_j = \sigma_j \mathbf{u}_j \quad 1 \leq j \leq n\,. \tag{15.1.4}$$

Thus in total, there are n vectors that are transformed under $\mathbf{A}$. A more compact way to write all of these equations simultaneously is with the representation

$$\begin{bmatrix} & & \\ & \mathbf{A} & \\ & & \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix} \tag{15.1.5}$$

so that in compact matrix notation this becomes

$$\mathbf{A}\mathbf{V} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\,. \tag{15.1.6}$$

The matrix $\hat{\mathbf{\Sigma}}$ is an $n \times n$ diagonal matrix with positive entries provided the matrix $\mathbf{A}$ is of full rank. The matrix $\hat{\mathbf{U}}$ is an $m \times n$ matrix with orthonormal columns, and the matrix $\mathbf{V}$ is an $n \times n$ unitary matrix. Since $\mathbf{V}$ is unitary, the above equation can be solved for $\mathbf{A}$ by multiplying on the right with $\mathbf{V}^*$ so that

$$\mathbf{A} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V}^*\,. \tag{15.1.7}$$

This factorization is known as the *reduced singular value decomposition*, or reduced SVD, of the matrix $\mathbf{A}$. Graphically, the factorization is represented in Fig. 15.3.

The reduced SVD is not the standard definition of the SVD used in the literature. What is typically done to augment the treatment above is to construct a matrix $\mathbf{U}$ from $\hat{\mathbf{U}}$ by adding an additional $m - n$ columns that are orthonormal to the already existing set in $\hat{\mathbf{U}}$. Thus the

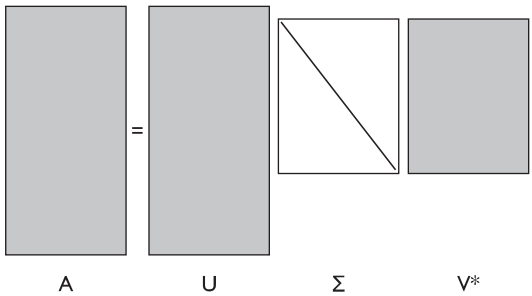

Figure 15.3: Graphical description of the reduced SVD decomposition.

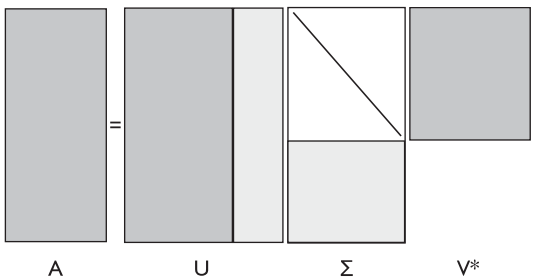

Figure 15.4: Graphical description of the full SVD decomposition where both **U** and **V** are unitary matrices. The light shaded regions of **U** and **Σ** are the silent rows and columns that are extended from the reduced SVD.

matrix **U** becomes an $m \times m$ unitary matrix. In order to make this procedure work, an additional $m - n$ rows of zeros is also added to the $\hat{\mathbf{\Sigma}}$ matrix. These "silent" columns of **U** and rows of **Σ** are shown graphically in Fig. 15.4. In performing this procedure, it becomes evident that rank deficient matrices can easily be handled by the SVD decomposition. In particular, instead of $m - n$ silent rows and matrices, there are now simply $m - r$ silent rows and columns added to the decomposition. Thus the matrix **Σ** will have r positive diagonal entries, with the remaining $n - r$ being equal to zero.

The full SVD decomposition thus take the form

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{15.1.8}$$

with the following three matrices

$$\mathbf{U} \in \mathbb{C}^{m\times m} \text{ is unitary} \tag{15.1.9a}$$
$$\mathbf{V} \in \mathbb{C}^{n\times n} \text{ is unitary} \tag{15.1.9b}$$
$$\mathbf{\Sigma} \in \mathbb{R}^{m\times n} \text{ is diagonal.} \tag{15.1.9c}$$

Additionally, it is assumed that the diagonal entries of Σ are nonnegative and ordered from largest to smallest so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ where $p = \min(m, n)$. The SVD decomposition of the matrix $\mathbf{A}$ thus shows that the matrix first applies a unitary transformation preserving the unit sphere via $\mathbf{V}^*$. This is followed by a stretching operation that creates an ellipse with principal semi-axes given by the matrix $\mathbf{\Sigma}$. Finally, the generated hyper-ellipse is rotated by the unitary transformation $\mathbf{U}$. Thus the statement: the image of a unit sphere under any $m \times n$ matrix is a hyper-ellipse, is shown to be true. The following is the primary theorem concerning the SVD.

Theorem *Every matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$ has a singular value decomposition (15.1.8). Furthermore, the singular values $\{\sigma_j\}$ are uniquely determined, and, if $\mathbf{A}$ is square and the σ_j distinct, the singular vectors $\{\mathbf{u}_j\}$ and $\{\mathbf{v}_j\}$ are uniquely determined up to complex signs (complex scalar factors of absolute value 1).*

Computing the SVD

The above theorem guarantees the existence of the SVD, but in practice, it still remains to be computed. This is a fairly straightforward process if one considers the following matrix products:

$$\begin{aligned}\mathbf{A}^T\mathbf{A} &= (\mathbf{U\Sigma V}^*)^T(\mathbf{U\Sigma V}^*)\\ &= \mathbf{V\Sigma U}^*\mathbf{U\Sigma V}^*\\ &= \mathbf{V\Sigma}^2\mathbf{V}^* \end{aligned} \tag{15.1.10}$$

and

$$\begin{aligned}\mathbf{AA}^T &= (\mathbf{U\Sigma V}^*)(\mathbf{U\Sigma V}^*)^T\\ &= \mathbf{U\Sigma V}^*\mathbf{V\Sigma U}^*\\ &= \mathbf{U\Sigma}^2\mathbf{U}^*\,. \end{aligned} \tag{15.1.11}$$

Multiplying (15.1.10) and (15.1.11) on the right by $\mathbf{V}$ and $\mathbf{U}$, respectively, gives the two self-consistent eigenvalue problems

$$\mathbf{A}^T\mathbf{AV} = \mathbf{V\Sigma}^2 \tag{15.1.12a}$$

$$\mathbf{AA}^T\mathbf{U} = \mathbf{U\Sigma}^2\,. \tag{15.1.12b}$$

Thus if the normalized eigenvectors are found for these two equations, then the orthonormal basis vectors are produced for $\mathbf{U}$ and $\mathbf{V}$. Likewise, the square root of the eigenvalues of these equations produces the singular values σ_j.

Example: Consider the SVD decomposition of

$$A = \begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix}. \tag{15.1.13}$$

The following quantities are computed:

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 4 \end{bmatrix} \tag{15.1.14a}$$

$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 4 \end{bmatrix}. \tag{15.1.14b}$$

The eigenvalues are clearly $\lambda = \{9, 4\}$, giving singular values $\sigma_1 = 3$ and $\sigma_2 = 2$. The eigenvectors can similarly be constructed and the matrices $\mathbf{U}$ and $\mathbf{V}$ are given by

$$\begin{bmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{bmatrix} \tag{15.1.15}$$

where there is an indeterminate sign in the eigenvectors. However, a self-consistent choice of signs must be made. One possible choice gives

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{15.1.16}$$

The SVD can be easily computed in MATLAB with the following command:

```
[U,S,V]=svd(A);
```

where the $[U, S, V]$ correspond to $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$, respectively. This decomposition is a critical tool for analyzing many data driven phenomena. But before proceeding to such applications, the connection of the SVD with standard and well-known techniques is elucidated.

15.2 The SVD in Broader Context

The SVD is an exceptionally important tool in many areas of applications. Part of this is due to its many mathematical properties and its guarantee of existence. In what follows, some of its more important theorems and relations to other standard ideas of linear algebra are considered with the aim of setting the mathematical framework for future applications.

Eigenvalues, eigenvectors and diagonalization

The concept of eigenvalues and eigenvectors is critical to understanding many areas of applications. One of the most important areas where it plays a role is in understanding differential equations. Consider, for instance, the system of differential equations:

$$\frac{d\mathbf{y}}{dt} = \mathbf{A}\mathbf{y} \tag{15.2.1}$$

for some vector $\mathbf{y}(t)$ representing a dynamical system of variables and where the matrix $\mathbf{A}$ determines the interaction among these variables. Assuming a solution of the form $\mathbf{y} = \mathbf{x}\exp(\lambda t)$ results in the eigenvalue problem:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}. \tag{15.2.2}$$

The question remains: How are the eigenvalues and eigenvectors found? To consider this problem, we rewrite the eigenvalue problem as

$$\mathbf{A}\mathbf{x} - \lambda\mathbf{I}\mathbf{x} = (\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}. \tag{15.2.3}$$

Two possibilities now exist.

Option I

The determinant of the matrix $(\mathbf{A} - \lambda\mathbf{I})$ is not zero. If this is true, the matrix is *nonsingular* and its inverse, $(\mathbf{A} - \lambda\mathbf{I})^{-1}$, can be found. The solution to the eigenvalue problem (15.2.2) is then

$$\mathbf{x} = (\mathbf{A} - \lambda\mathbf{I})^{-1}\mathbf{0} \tag{15.2.4}$$

which implies that $\mathbf{x} = \mathbf{0}$. This trivial solution could have easily been guessed. However, it is not relevant as we require nontrivial solutions for $\mathbf{x}$.

Option II

The determinant of the matrix $(\mathbf{A} - \lambda\mathbf{I})$ is zero. If this is true, the matrix is *singular* and its inverse, $(\mathbf{A} - \lambda\mathbf{I})^{-1}$, cannot be found. Although there is no longer a guarantee that there is a solution, it is the only scenario which allows for the possibility of $\mathbf{x} \neq 0$. It is this condition which allows for the construction of eigenvalues and eigenvectors. Indeed, we choose the eigenvalues λ so that this condition holds and the matrix is singular.

Another important operation which can be performed with eigenvalues and eigenvectors is the evaluation of

$$\mathbf{A}^M \tag{15.2.5}$$

where M is a large integer. For large matrices $\mathbf{A}$, this operation is computationally expensive. However, knowing the eigenvalues and eigenvectors of $\mathbf{A}$ allows for a significant ease in computational expense. Assuming we have all the eigenvalues and eigenvectors of $\mathbf{A}$, then

$$\begin{aligned}
\mathbf{A}\mathbf{x}_1 &= \lambda_1\mathbf{x}_1 \\
\mathbf{A}\mathbf{x}_2 &= \lambda_2\mathbf{x}_2 \\
&\vdots \\
\mathbf{A}\mathbf{x}_n &= \lambda_1\mathbf{x}_n\,.
\end{aligned}$$

This collection of eigenvalues and eigenvectors gives the matrix system

$$\mathbf{A}\mathbf{S} = \mathbf{S}\boldsymbol{\Lambda} \tag{15.2.6}$$

where the columns of the matrix $\mathbf{S}$ are the eigenvectors of $\mathbf{A}$,

$$\mathbf{S} = (\mathbf{x}_1 \ \mathbf{x}_2 \ \cdots \ \mathbf{x}_n) \,, \tag{15.2.7}$$

and $\mathbf{\Lambda}$ is a matrix whose diagonals are the corresponding eigenvalues

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & \cdots & 0 & \lambda_n \end{pmatrix} . \tag{15.2.8}$$

By multiplying (15.2.6) on the right by $\mathbf{S}^{-1}$, the matrix $\mathbf{A}$ can then be rewritten as (note the similarity between this expression and Eq. (15.1.8) for the SVD decomposition)

$$\mathbf{A} = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1} . \tag{15.2.9}$$

The final observation comes from

$$\mathbf{A}^2 = (\mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1})(\mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1}) = \mathbf{S}\mathbf{\Lambda}^2\mathbf{S}^{-1} . \tag{15.2.10}$$

This then generalizes to

$$\mathbf{A}^M = \mathbf{S}\mathbf{\Lambda}^M\mathbf{S}^{-1} \tag{15.2.11}$$

where the matrix $\mathbf{\Lambda}^M$ is easily calculated as

$$\mathbf{\Lambda}^M = \begin{pmatrix} \lambda_1^M & 0 & \cdots & & 0 \\ 0 & \lambda_2^M & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & \cdots & 0 & \lambda_n^M \end{pmatrix} . \tag{15.2.12}$$

Since raising the diagonal terms to the Mth power is easily accomplished, the matrix $\mathbf{A}$ can then be easily calculated by multiplying the three matrices in (15.2.11).

Diagonalization can also recast a given problem so as to elucidate its more fundamental dynamics. A classic example of the use of diagonalization is a two-mass spring system where the masses m_1 and m_2 are acted on by forces $F_1(t)$ and $F_2(t)$. A schematic of this situation is depicted in Fig. 15.5. For each mass, we can write down Newton's law:

$$\sum F_1 = m_1 \frac{d^2x_1}{dt^2} \quad \text{and} \quad \sum F_2 = m_2 \frac{d^2x_2}{dt^2} \tag{15.2.13}$$

where $\sum F_1$ and $\sum F_2$ are the sums of the forces on m_1 and m_2, respectively. Note that the equations for $x_1(t)$ and $x_2(t)$ are coupled because of the spring with spring constant k_2. The resulting governing equations are then of the form:

$$m_1 \frac{d^2x_1}{dt^2} = -k_1x_1 + k_2(x_2 - x_1) + F_1 = -(k_1 + k_2)x_1 + k_2x_2 + F_1 \tag{15.2.14a}$$

$$m_2 \frac{d^2x_2}{dt^2} = -k_3x_2 - k_2(x_2 - x_1) + F_2 = -(k_2 + k_3)x_2 + k_2x_1 + F_2. \tag{15.2.14b}$$

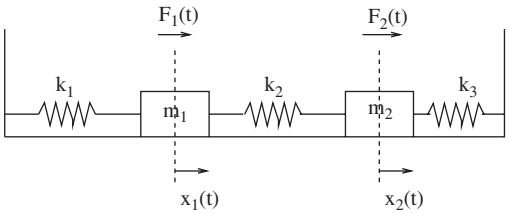

Figure 15.5: A two-mass, three-spring system. The fundamental behavior of the system can be understood by decomposition of the system via diagonalization. This reveals that all motion can be expressed as the sum of two fundamental motions: the masses oscillating in-phase, and the masses oscillating exactly out-of-phase.

This can be reduced further by assuming, for simplicity, $m = m_1 = m_2$, $K = k_1/m = k_2/m$ and $F_1 = F_2 = 0$. This results in the linear system which can be diagonalized via Eq. (15.2.9). The pairs of complex conjugate eigenvalues are produced: $\lambda_1^{\pm} = \pm i(2K + \sqrt{2K})^{1/2}$ and $\lambda_2^{\pm} = \pm i(2K - \sqrt{2K})^{1/2}$. Upon diagonalization, the full system can be understood as simply a linear combination of oscillations of the masses that are in-phase with each other or out-of-phase with each other.

Diagonalization via SVD

Like the eigenvalue and eigenvector diagonalization technique presented above, the SVD method also makes the following claim: *the SVD makes it possible for every matrix to be diagonal if the proper bases for the domain and range are used.* To consider this statement more formally, consider that since $\mathbf{U}$ and $\mathbf{V}$ are orthonormal bases in $\mathbb{C}^{m\times m}$ and $\mathbb{C}^{n\times n}$, respectively, then any vector in these spaces can be expanded in their bases. Specifically, consider a vector $\mathbf{b} \in \mathbb{C}^{m\times m}$ and $\mathbf{x} \in \mathbb{C}^{n\times n}$. Then each can be expanded in the bases of $\mathbf{U}$ and $\mathbf{V}$ so that

$$\mathbf{b} = \mathbf{U}\hat{\mathbf{b}}, \quad \mathbf{x} = \mathbf{V}\hat{\mathbf{x}} \tag{15.2.15}$$

where the vectors $\hat{\mathbf{b}}$ and $\hat{\mathbf{x}}$ give the weightings for the orthonormal bases expansion. Now consider the simple equation:

$$\begin{aligned} \mathbf{A}\mathbf{x} = \mathbf{b} \rightarrow \ & \mathbf{U}^*\mathbf{b} = \mathbf{U}^*\mathbf{A}\mathbf{x} \\ & \mathbf{U}^*\mathbf{b} = \mathbf{U}^*\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*\mathbf{x} \\ & \hat{\mathbf{b}} = \boldsymbol{\Sigma}\hat{\mathbf{x}}. \end{aligned} \tag{15.2.16}$$

Thus the last line shows that $\mathbf{A}$ reduces to the diagonal matrix $\boldsymbol{\Sigma}$ when the range is expressed in terms of the basis vectors of $\mathbf{U}$ and the domain is expressed in terms of the basis vectors of $\mathbf{V}$.

Thus matrices can be diagonalized via either an eigenvalue decomposition or an SVD decomposition. However, there are three key differences in the diagonalization process.

- The SVD performs the diagonalization using two different bases, $\mathbf{U}$ and $\mathbf{V}$, while the eigenvalue method uses a single basis $\mathbf{X}$.

- The SVD method uses an orthonormal basis while the basis vectors in $\mathbf{X}$, while linearly independent, are not generally orthogonal.
- Finally, the SVD is guaranteed to exist for any matrix $\mathbf{A}$ while the same is not true, even for square matrices, for the eigenvalue decomposition.

Useful theorems of SVD

Having established the similarities and connections between eigenvalues and singular values, in what follows a number of important theorems are outlined concerning the SVD. These theorems are important for several reasons. First, the theorems play a key role in the numerical evaluation of many matrix properties via the SVD. Second, the theorems guarantee certain behaviors of the SVD that can be capitalized upon for future applications. Here is a list of important results. A more detailed account is given by Trefethen and Bau [30].

Theorem *If the rank of* $\mathbf{A}$ *is r, then there are r nonzero singular values.*

The proof of this is based upon the fact that the rank of a diagonal matrix is equal to the number of its nonzero entries. And because of the decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ where $\mathbf{U}$ and $\mathbf{V}$ are full rank, then $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{\Sigma}) = r$. As a side note, the rank of a matrix can be found with the MATLAB command:

```
rank(A)
```

The standard algorithm used to compute the rank is based upon the SVD and the computation of the nonzero singular values. Thus the theorem is quite useful in practice.

Theorem *range(*$\mathbf{A}$*)=*$\langle \mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_r \rangle$ *and null (*$\mathbf{A}$*)=*$\langle \mathbf{v}_{r+1}, \mathbf{v}_{r+2}, \cdots, \mathbf{v}_n \rangle$.

Note that the range and null space come from the two expansion bases $\mathbf{U}$ and $\mathbf{V}$. The range and null space can be found in MATLAB with the commands:

```
range(A)
null(A)
```

This theorem also serves as the most accurate computation basis for determining the range and null space of a given matrix $\mathbf{A}$ via the SVD.

Theorem $\|\mathbf{A}\|_2 = \sigma_1$ *and* $\|\mathbf{A}\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2}$.

These norms are known as the 2-norm and the Frobenius norm, respectively. They essentially measure the *energy* of a matrix. Although it is difficult to conceptualize abstractly what this means, it will become much more clear in the context of given applications. This result is established by the fact that $\mathbf{U}$ and $\mathbf{V}$ are unitary operators so that their norm is unity. Thus with $\mathbf{A} = \mathbf{U\Sigma V}^*$, the 2-norm is $\|\mathbf{A}\|_2 = \|\mathbf{\Sigma}\|_2 = \max\{|\sigma_j|\} = \sigma_1$. Similar reasoning holds for the Frobenius norm definition. The norm can be calculated in MATLAB with

```
norm(A)
```

Notice that the Frobenius norm contains the total matrix energy while the 2-norm definition contains the energy of the largest singular value. The ratio $\|\mathbf{A}\|_2/\|\mathbf{A}\|_F$ effectively measures the portion of the energy in the semi-axis $\mathbf{u}_1$. This fact will be tremendously important for us. Furthermore, this theorem gives the standard way for computing matrix norms.

Theorem *The nonzero singular values of* $\mathbf{A}$ *are the square roots of the nonzero eigenvalues of* $\mathbf{A}^*\mathbf{A}$ *or* $\mathbf{AA}^*$. *(These matrices have the same nonzero eigenvalues.)*

This has already been shown in the calculation for actually determining the SVD. In particular, this process is illustrated in Eqs. (15.1.11) and (15.1.10). This theorem is important for actually producing the unitary matrices $\mathbf{U}$ and $\mathbf{V}$.

Theorem *If* $\mathbf{A} = \mathbf{A}^*$ *(self-adjoint), then the singular values of* $\mathbf{A}$ *are the absolute values of the eigenvalues of* $\mathbf{A}$.

As with most self-adjoint problems, there are very nice properties of the matrices, such as the above theorem. Eigenvalues can be computed with the command:

```
eig(A)
```

Alternatively, one can use the **eigs** to specify the number of eigenvalues desired and their specific ordering.

Theorem *For* $\mathbf{A} \in \mathbb{C}^{m\times m}$, *the determinant is given by* $|\det(A)| = \prod_{j=1}^{m} \sigma_j$.

Again, due to the fact that the matrices $\mathbf{U}$ and $\mathbf{V}$ are unitary, their determinants are unity. Thus $|\det(A)| = |\det(\mathbf{U\Sigma V}^*)| = |\det(\mathbf{U})||\det(\mathbf{\Sigma})||\det(\mathbf{V}^*)| = |\det(\mathbf{\Sigma})| = \prod_{j=1}^{m} \sigma_j$. Thus even determinants can be computed via the SVD and its singular values.

Low dimensional reductions

Now comes the last, and most formative, property associated with the SVD: *low dimensional approximations* to high degree of freedom or complex systems. In linear algebra terms, this is also *low rank approximations*. The interpretation of the theorems associated with these low dimensional reductions are critical for the use and implementation of the SVD. Thus we consider the following:

Theorem A *is the sum of r rank-one matrices*

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^* \,. \tag{15.2.17}$$

There are a variety of ways to express the $m \times n$ matrix $\mathbf{A}$ as a sum of rank-one matrices. The bottom line is this: *the Nth partial sum captures as much of the matrix* $\mathbf{A}$ *as possible*. Thus the partial sum of the rank-one matrices is an important object to consider. This leads to the following theorem:

Theorem *For any N so that* $0 \leq N \leq r$*, we can define the partial sum*

$$\mathbf{A}_N = \sum_{j=1}^{N} \sigma_j \mathbf{u}_j \mathbf{v}_j^* \,. \tag{15.2.18}$$

And if $N = \min\{m, n\}$*, define* $\sigma_{N+1} = 0$*. Then*

$$\|A - A_N\|_2 = \sigma_{N+1} \,. \tag{15.2.19}$$

Likewise, if using the Frobenius norm, then

$$\|A - A_N\|_N = \sqrt{\sigma_{N+1}^2 + \sigma_{N+2}^2 + \cdots + \sigma_r^2} \,. \tag{15.2.20}$$

Interpreting this theorem is critical. Geometrically, we can ask *what is the best approximation of a hyper-ellipsoid by a line segment? Simply take the line segment to be the longest axis, i.e. that associated with the singular value* σ_1. Continuing this idea, what is the best approximation by a two-dimensional ellipse? Take the longest and second longest axes, i.e. those associated with the singular values σ_1 and σ_2. After r steps, the total energy in $\mathbf{A}$ is completely captured. **Thus the SVD gives a type of least-square fitting algorithm, allowing us to project the matrix onto low dimensional representations in a formal, algorithmic way.** Herein lies the ultimate power of the method.

15.3 Introduction to Principal Component Analysis (PCA)

To make explicit the concept of the SVD, a simple model example will be formulated that will illustrate all the key concepts associated with the SVD. The model to be considered will be a

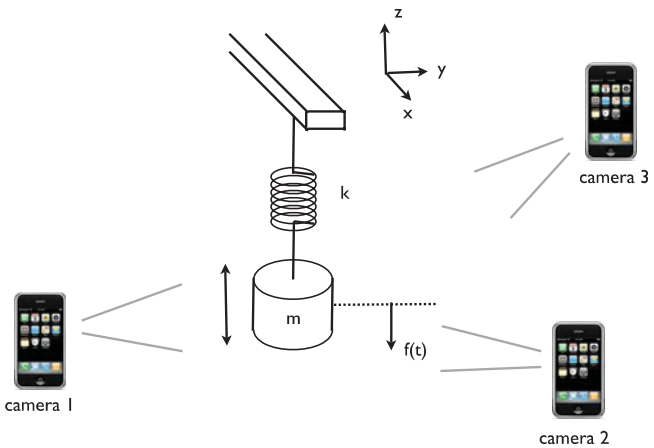

Figure 15.6: A prototypical example of how we might apply a principal component analysis, or SVD, is the simple mass–spring system exhibited here. The mass m is suspended with a spring with Hooke's constant k. Three video cameras collect data about its motion in order to ascertain its governing equations.

simple spring–mass system as illustrated in Fig. 15.6. Of course, this is a fairly easy problem to solve from basic concepts of $\mathbf{F} = m\mathbf{a}$. But for the moment, let's suppose we didn't know the governing equations. In fact, our aim in this section is to use a number of cameras (probes) to extract out data concerning the behavior of the system and then to extract empirically the governing equations of motion.

This prologue highlights one of the key applications of the SVD, or alternatively a variant of *principal component analysis* (PCA), *proper mode decomposition* (POD), *Hotelling transform*, *empirical orthogonal functions* (EOF), *reduced order modeling* (ROM), *dimensionality reduction* or *Karhunen–Loéve decomposition* as it is also known in the literature. Namely, from seemingly complex, perhaps random data, can low dimensional reductions of the dynamics and behavior be produced when the governing equations are not known? Such methods can be used to quantify low dimensional dynamics arising in such areas as turbulent fluid flows [32], structural vibrations [33, 34], insect locomotion [35], damage detection [36], and neural decision making strategies [37], to name just a few areas of application. It will also become obvious as we move forward on this topic that the breadth of applications is staggering and includes image processing and signal analysis. Thus the perspective to be taken here is clearly one in which the data analysis of an unknown, but potentially low dimensional system is to be analyzed.

Again we turn our attention to the simple experiment at hand: a mass suspended by a spring as depicted in Fig. 15.6. If the mass is perturbed or taken from equilibrium in the z-direction only, we know that the governing equations are simply

$$\frac{d^2f(t)}{dt^2} = -\omega^2 f(t) \tag{15.3.1}$$

where the function $f(t)$ measures the displacement of the mass in the z-direction as a function of time. This has the well-known solution (in amplitude–phase form)

$$f(t) = A\cos(\omega t + \omega_0) \tag{15.3.2}$$

where the values of A and ω_0 are determined from the initial state of the system. This essentially states that the state of the system can be described by a one degree of freedom system.

In the above analysis, there are many things we have ignored. Included in the list of things we have ignored is the possibility that the initial excitation of the system actually produces movement in the $x-y$ plane. Further, there is potentially noise in the data from, for instance, shaking of the cameras during the video capture. Moreover, from what we know of the solution, only a single camera is necessary to capture the underlying motion. In particular, a single camera in the $x-y$ plane at $z = 0$ would be ideal. Thus we have oversampled the data with three cameras and have produced redundant data sets. From all of these potential perturbations and problems, it is our goal to extract out the simple solution given by the simple harmonic motion.

This problem is a good example of what kind of processing is required to analyze a realistic data set. Indeed, one can imagine that most data will be quite noisy, perhaps redundant, and certainly not produced from an optimal viewpoint. But through the process of PCA, these can be circumvented in order to extract out the ideal or simplified behavior. Moreover, we may even learn how to transform the data into the optimal viewpoint for analyzing the data.

Data collection and ordering

Assume now that we have started the mass in motion by applying a small perturbation in the z-direction only. Thus the mass will begin to oscillate. Three cameras are then used to record the motion. Each camera produces a two-dimensional representation of the data. If we denote the data from the three cameras with subscripts a, b and c, then the data collected are represented by the following:

$$\text{camera 1: } (\mathbf{x}_a, \mathbf{y}_a) \tag{15.3.3a}$$

$$\text{camera 2: } (\mathbf{x}_b, \mathbf{y}_b) \tag{15.3.3b}$$

$$\text{camera 3: } (\mathbf{x}_c, \mathbf{y}_c) \tag{15.3.3c}$$

where each set $(\mathbf{x}_j, \mathbf{y}_j)$ is data collected over time of the position in the $x-y$ plane of the camera. Note that this is not the same $x-y$ plane of the oscillating mass system as shown in Fig. 15.6. Indeed, we should pretend we don't know the correct $x-y-z$ coordinates of the system. Thus the camera positions and their relative $x-y$ planes are arbitrary. The length of each vector $\mathbf{x}_i$ and $\mathbf{y}_i$ depends on the data collection rate and the length of time the dynamics is observed. We denote the length of these vectors as n.

All the data collected can then be gathered into a single matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_a \\ \mathbf{y}_a \\ \mathbf{x}_b \\ \mathbf{y}_b \\ \mathbf{x}_c \\ \mathbf{y}_c \end{bmatrix}. \tag{15.3.4}$$

Thus the matrix $X \in \mathbb{R}^{m\times n}$ where m represents the number of measurement types and n is the number of data points taken from the camera over time.

Now that the data have been arranged, two issues must be addressed: noise and redundancy. Everyone has an intuitive concept that noise in your data can only deteriorate, or corrupt, your ability to extract the true dynamics. Just as in image processing, noise can alter an image beyond restoration. Thus there is also some idea that if the measured data are too noisy, fidelity of the underlying dynamics is compromised from a data analysis point of view. The key measure of this is the so-called signal-to-noise ratio: $\text{SNR} = \sigma^2_{\text{signal}}/\sigma^2_{\text{noise}}$, where the ratio is given as the ratio of the variances of the signal and noise fields. A high SNR (much greater than unity) gives almost noiseless (high precision) data whereas a low SNR suggests the underlying signal is corrupted by the noise. The second issue to consider is redundancy. In the example of Fig. 15.6, the single degree of freedom is sampled by three cameras, each of which is really recording the same single degree of freedom. Thus the measurements should be rife with redundancy, suggesting that the different measurements are statistically dependent. Removing this redundancy is critical for data analysis.

The covariance matrix

An easy way to identify redundant data is by considering the covariance between data sets. Recall from our early chapters on probability and statistics that the covariance measures the statistical dependence/independence between two variables. Obviously, strongly statistically dependent variables can be considered as redundant observations of the system. Specifically, consider two sets of measurements with zero means expressed in row vector form:

$$\mathbf{a} = [a_1\ a_2\ \cdots\ a_n] \quad \text{and} \quad \mathbf{b} = [b_1\ b_2\ \cdots\ b_n] \tag{15.3.5}$$

where the subscript denotes the sample number. The variances of $\mathbf{a}$ and $\mathbf{b}$ are given by

$$\sigma^2_{\mathbf{a}} = \frac{1}{n-1}\mathbf{a}\mathbf{a}^T \tag{15.3.6a}$$

$$\sigma^2_{\mathbf{b}} = \frac{1}{n-1}\mathbf{b}\mathbf{b}^T \tag{15.3.6b}$$

while the covariance between these two data sets is given by

$$\sigma^2_{\mathbf{ab}} = \frac{1}{n-1}\mathbf{a}\mathbf{b}^T \tag{15.3.7}$$

where the normalization constant of $1/(n-1)$ is for an unbiased estimator.

We don't just have two vectors, but potentially quite a number of experiments and data that would need to be correlated and checked for redundancy. In fact, the matrix in Eq. (15.3.4) is exactly what needs to be checked for covariance. The appropriate *covariance matrix* for this case is then

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T. \tag{15.3.8}$$

This is easily computed with MATLAB from the command line:

```
cov(X)
```

The covariance matrix $\mathbf{C_X}$ is a square, symmetric $m \times m$ matrix whose diagonal represents the variance of particular measurements. The off-diagonal terms are the covariances between measurement types. Thus $\mathbf{C_X}$ captures the correlations between all possible pairs of measurements. Redundancy is thus easily captured since if two data sets are identical (identically redundant), the off-diagonal term and diagonal term would be equal since $\sigma^2_{\mathbf{ab}} = \sigma^2_{\mathbf{a}} = \sigma^2_{\mathbf{b}}$ if $\mathbf{a} = \mathbf{b}$. Thus large off-diagonal terms correspond to redundancy while small off-diagonal terms suggest that the two measured quantities are close to being statistically independent and have low redundancy. It should also be noted that large diagonal terms, or those with large variances, typically represent what we might consider *the dynamics of interest* since the large variance suggests strong fluctuations in that variable. Thus the covariance matrix is the key component to understanding the entire data analysis.

Achieving the goal

The insight given by the covariance matrix leads to our ultimate aim of

(i) removing redundancy

(ii) identifying those signals with maximal variance.

Thus in a mathematical sense we are simply asking to represent $\mathbf{C_X}$ so that the diagonals are ordered from largest to smallest and the off-diagonals are zero, i.e. our task is to diagonalize the covariance matrix. This is *exactly* what the SVD does, thus allowing it to becomes the tool of choice for data analysis and dimensional reduction. In fact, the SVD diagonalizes and each singular direction captures as much energy as possible as measured by the singular values σ_j.

15.4 Principal Components, Diagonalization and SVD

The example presented in the previous section shows that the key to analyzing a given experiment is to consider the covariance matrix

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \tag{15.4.1}$$

where the matrix $\mathbf{X}$ contains the experimental data of the system. In particular, $\mathbf{X} \in \mathbb{C}^{m \times n}$ where m are the number of probes or measuring positions, and n is the number of experimental data points taken at each location.

In this setup, the following facts are highlighted:

- $\mathbf{C_X}$ is a square, symmetric $m \times m$ matrix.
- The diagonal terms of $\mathbf{C_X}$ are the variances for particular measurements. By assumption, large variances correspond to *dynamics of interest*, whereas low variances are assumed to correspond to *uninteresting dynamics*.
- The off-diagonal terms of $\mathbf{C_X}$ are the covariances between measurements. Indeed, the off-diagonals capture the correlations between all possible pairs of measurements. A large off-diagonal term represents two events that have a high degree of redundancy, whereas a small off-diagonal coefficient means there is little redundancy in the data, i.e. they are statistically independent.

Diagonalization

The concept of diagonalization is critical for understanding the underpinnings of many physical systems. In this process of diagonalization, the correct coordinates, or basis functions, are revealed that reduce the given system to its low dimensional essence. There is more than one way to diagonalize a matrix, and this is certainly true here as well since the constructed covariance matrix $\mathbf{C_X}$ is square and symmetric, both properties that are especially beneficial for standard eigenvalue/eigenvector expansion techniques.

The key idea behind the diagonalization is simply this: there exists an *ideal* basis in which the $\mathbf{C_X}$ can be written (diagonalized) so that in this basis, all redundancies have been removed, and the largest variances of particular measurements are ordered. In the language being developed here, this means that the system has been written in terms of its *principal components*, or in a *proper orthogonal decomposition*.

Eigenvectors and eigenvalues

The most straightforward way to diagonalize the covariance matrix is by making the observation that $\mathbf{XX}^T$ is a square, symmetric $m \times m$ matrix, i.e. it is self-adjoint so that the m eigenvalues are real and distinct. Linear algebra provides theorems which state that such a matrix can be rewritten as

$$\mathbf{XX}^T = \mathbf{S}\boldsymbol{\Lambda}\mathbf{S}^{-1} \tag{15.4.2}$$

as stated in Eq. (15.2.9) where the matrix $\mathbf{S}$ is a matrix of the eigenvectors of $\mathbf{XX}^T$ arranged in columns. Since it is a symmetric matrix, these eigenvector columns are orthogonal so that ultimately the $\mathbf{S}$ can be written as a unitary matrix with $\mathbf{S}^{-1} = \mathbf{S}^T$. Recall that the matrix $\boldsymbol{\Lambda}$ is a diagonal matrix whose entries correspond to the m distinct eigenvalue of $\mathbf{XX}^T$.

This suggests that instead of working directly with the matrix $\mathbf{X}$, we consider working with the transformed variable, or in the principal component basis,

$$\mathbf{Y} = \mathbf{S}^T\mathbf{X}. \tag{15.4.3}$$

For this new basis, we can then consider its covariance

$$\begin{aligned}\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T \\ &= \frac{1}{n-1}(\mathbf{S}^T\mathbf{X})(\mathbf{S}^T\mathbf{X})^T \\ &= \frac{1}{n-1}\mathbf{S}^T(\mathbf{X}\mathbf{X}^T)\mathbf{S} \\ &= \frac{1}{n-1}\mathbf{S}^T\mathbf{S}\mathbf{\Lambda}\mathbf{S}\mathbf{S}^T \\ \mathbf{C_Y} &= \frac{1}{n-1}\mathbf{\Lambda}\,. \end{aligned} \tag{15.4.4}$$

In this basis, the *principal components* are the eigenvectors of $\mathbf{X}\mathbf{X}^T$ with the interpretation that the jth diagonal value of $\mathbf{C_Y}$ is the variance of $\mathbf{X}$ along $\mathbf{x}_j$, the jth column of $\mathbf{S}$. The following lines of code produce the principal components of interest.

```
[m,n]=size(X);   %  compute data size
mn=mean(X,2); %  compute mean for each row
X=X-repmat(mn,1,n);  % subtract mean

Cx=(1/(n-1))*X*X';   % covariance
[V,D]=eig(Cx);      % eigenvectors(V)/eigenvalues(D)
lambda=diag(D);    % get eigenvalues

[dummy,m_arrange]=sort(-1*lambda); % sort in decreasing order
lambda=lambda(m_arrange);
V=V(:,m_arrange);

Y=V'*X;  % produce the principal components projection
```

This simple code thus produces the eigenvalue decomposition and the projection of the original data onto the principal component basis.

Singular value decomposition

A second method for diagonalizing the covariance matrix is the SVD method. In this case, the SVD can diagonalize any matrix by working in the appropriate pair of bases $\mathbf{U}$ and $\mathbf{V}$ as outlined in the first part of this chapter. Thus by defining the transformed variable

$$\mathbf{Y} = \mathbf{U}^*\mathbf{X} \tag{15.4.5}$$

where $\mathbf{U}$ is the unitary transformation associated with the SVD: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$. Just as in the eigenvalue/eigenvector formulation, we then compute the variance in $\mathbf{Y}$:

$$\begin{aligned}\mathbf{C}_\mathbf{Y} &= \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T \\ &= \frac{1}{n-1}(\mathbf{U}^*\mathbf{X})(\mathbf{U}^*\mathbf{X})^T \\ &= \frac{1}{n-1}\mathbf{U}^*(\mathbf{X}\mathbf{X}^T)\mathbf{U} \\ &= \frac{1}{n-1}\mathbf{U}^*\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}\mathbf{U}^* \\ \mathbf{C}_\mathbf{Y} &= \frac{1}{n-1}\mathbf{\Sigma}^2 . \end{aligned} \tag{15.4.6}$$

This makes explicit the connection between the SVD and the eigenvalue method, namely that $\mathbf{\Sigma}^2 = \mathbf{\Lambda}$. The following lines of code produce the principal components of interest using the SVD (assume that you have the first three lines from the previous MATLAB code).

```
[u,s,v]=svd(X'/sqrt(n-1));  % perform the SVD
lambda=diag(s).^2;  % produce diagonal variances
Y=u'*X;  % produce the principal components projection
```

This gives the SVD method for producing the principal components. Overall, the SVD method is the more robust method and should be used. However, the connection between the two methods becomes apparent in these calculations.

Spring experiment

To illustrate this completely in practice, three experiments will be performed with the configuration of Fig. 15.6. The following experiments will attempt to illustrate various aspects of the PCA and its practical usefulness and the effects of noise on the PCA algorithms.

- **Ideal case** Consider a small displacement of the mass in the z-direction and the ensuing oscillations. In this case, the entire motion is in the z-direction with simple harmonic motion being observed.
- **Noisy case** Repeat the ideal case experiment, but this time, introduce camera shake into the video recording. This should make it more difficult to extract the simple harmonic motion. But if the shake isn't too bad, the dynamics will still be extracted with the PCA algorithms.
- **Horizontal displacement** In this case, the mass is released off-center so as to produce motion in the $x-y$ plane as well as the z-direction. Thus there is both a pendulum motion and a simple harmonic oscillation. See what the PCA tells us about the system.

In order to extract out the mass movement from the video frames, the following MATLAB code is needed. This code is a generic way to read in movie files to MATLAB for post-processing.

```
obj=mmreader('matlab_test.mov')

vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');

for k = 1 : numFrames
    mov(k).cdata = vidFrames(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames
  X=frame2im(mov(j));
  imshow(X); drawnow
end
```

This multimedia reader command **mmreader** simply characterizes the file type and its attributes. The bulk of time in this is the **read** command which actually uploads the movie frames into the MATLAB desktop for processing. Once the frames are extracted, they can again be converted to double precision numbers for mathematical processing. In this case, the position of the mass is to be determined from each frame. This basic shell of code is enough to begin the process of extracting the spring–mass system information.

15.5 Principal Components and Proper Orthogonal Modes

Now that the basic framework of the principal component analysis and its relation to the SVD has been laid down, a few remaining issues need to be addressed. In particular, the principal component analysis seems to suggest that we are simply expanding our solution in another *orthonormal* basis, one which can always diagonalize the underlying system.

Mathematically, we can consider a given function $f(x,t)$ over a domain of interest. In most applications, the variables x and t will refer to the standard space–time variables we are familiar with. The idea is to then expand this function in some basis representation so that

$$f(x,t) \approx \sum_{j=1}^{N} a_j(t)\phi_j(x) \tag{15.5.1}$$

where N is the total number of modes, $\phi_j(x)$, to be used. The remaining function $a_j(t)$ determines the weights of the spatial modes.

The expansion (15.5.1) is certainly not a new idea to us. Indeed, we have been using this concept extensively already with Fourier transforms, for instance. Specifically, here are some of the more common expansion bases used in practice:

$$\phi_j(x) = (x - x_0)^j \quad \text{Taylor expansion} \tag{15.5.2a}$$
$$\phi_j(x) = \cos(jx) \quad \text{Discrete cosine transform} \tag{15.5.2b}$$
$$\phi_j(x) = \sin(jx) \quad \text{Discrete sine transform} \tag{15.5.2c}$$
$$\phi_j(x) = \exp(jx) \quad \text{Fourier transform} \tag{15.5.2d}$$
$$\phi_j(x) = \psi_{a,b}(x) \quad \text{Wavelet transform} \tag{15.5.2e}$$
$$\phi_j(x) = \phi_{\lambda_j}(x) \quad \text{Eigenfunction expansion} \tag{15.5.2f}$$

where $\phi_{\lambda_j}(x)$ are eigenfunctions associated with the underlying system. This places the concept of a basis function expansion in familiar territory. Further, it shows that such a basis function expansion is not unique, but rather can potentially have an infinite number of different possibilities.

As for the weighting coefficients $a_j(t)$, they are simply determined from the standard inner product rules and the fact that the basis functions are orthonormal (note that they are not written this way above):

$$\int \phi_j(x)\phi_n(x)dx = \begin{cases} 1 & j = n \\ 0 & j \neq n. \end{cases} \tag{15.5.3}$$

This then gives for the coefficients

$$a_j(t) = \int f(x,t)\phi_j(x)dx \tag{15.5.4}$$

and the basis expansion is completed.

Interestingly enough, the basis functions selected are often chosen for simplicity and/or their intuitive meaning for the system. For instance, the Fourier transform very clearly highlights the fact that the given function has a representation in terms of *frequency* components. This is fundamental for understanding many physical problems as there is clear and intuitive meaning to the Fourier modes.

In contrast to selecting basis functions for simplicity or intuitive meaning, the broader question can be asked: what criteria should be used for selecting the functions $\phi_j(x)$? This is an interesting question given that any complete basis can approximate the function $f(x,t)$ to any desired accuracy given N large enough. But what is desired is the best basis functions such that, with N as small as possible, we achieve the desired accuracy. The goal is then the following: find a sequence of orthonormal basis functions $\phi_j(x)$ so that the first two terms give the best two-term approximation to $f(x,t)$, or the first five terms give the best five-term approximation to $f(x,t)$. These special, ordered, orthonormal functions are called the *proper orthogonal modes* (POD) for the function $f(x,t)$. With these modes, the expansion (15.5.4) is called the POD of $f(x,t)$. The relation to the SVD will become obvious momentarily.

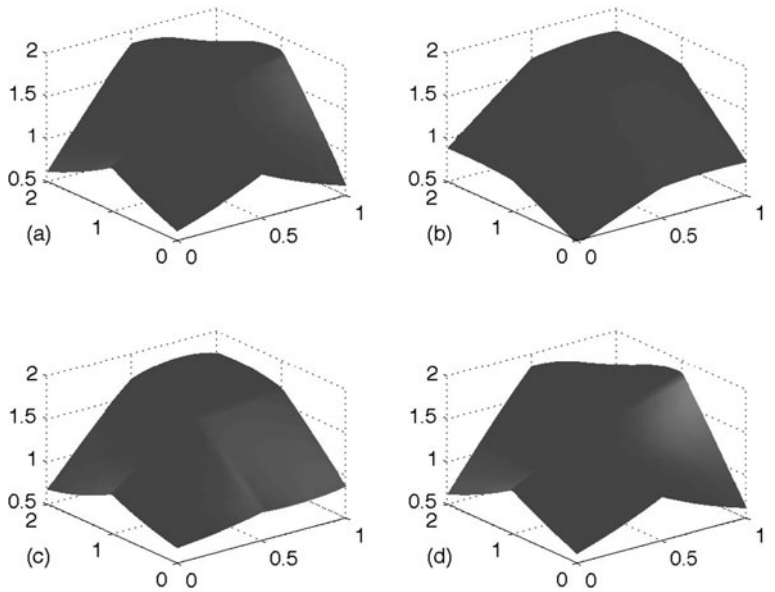

Figure 15.7: Representation of the surface (a) by a series of low rank (low-dimensional) approximations with (b) one mode, (c) two modes and (d) three modes. The energy captured in the first mode is approximately 92% of the total surface energy. The first three modes together capture 99.9% of the total surface energy, thus suggesting that the surface can be easily and accurately represented by a three-mode expansion.

Example 1: Consider an approximation to the following surface

$$f(x,t) = e^{-|(x-0.5)(t-1)|} + \sin(xt) \quad x \in [0,1],\ t \in [0,2]. \tag{15.5.5}$$

As listed above, there are a number of methods available for approximating this function using various basis functions. And all of the ones listed are guaranteed to converge to the surface for a large enough N in (15.5.1).

In the POD method, the surface is first discretized and approximated by a finite number of points. In particular, the following discretization will be used:

```
x=linspace(0,1,25);
t=linspace(0,2,50);
```

where x has been discretized into 25 points and t is discretized into 50 points. The surface is represented in Fig. 15.7(a) over the domain of interest. The surface is constructed by using the **meshgrid** command as follows:

```
[T,X]=meshgrid(t,x);
f=exp(-abs((X-.5).*(T-1)))+sin(X.*T);
subplot(2,2,1)
surfl(X,T,f), shading interp, colormap(gray)
```

The **surfl** produces a lighted surface that in this case is represented in gray-scale. Note that in producing this surface, the matrix **f** is a 25×50 matrix so that the x-values are given as row locations while the t-values are given by column locations.

The basis functions are then computed using the SVD. Recall that the SVD produces ordered singular values and associated orthonormal basis functions that capture as much energy as possible. Thus an SVD can be performed on the matrix **f** that represents the surface.

```
[u,s,v]=svd(f);  % perform SVD

for j=1:3
   ff=u(:,1:j)*s(1:j,1:j)*v(:,1:j)'; % modal projections
   subplot(2,2,j+1)
   surfl(X,T,ff), shading interp, colormap(gray)
   set(gca,'Zlim',[0.5 2])
end
subplot(2,2,1), text(-0.5,1,0.5,'(a)','Fontsize',[14])
subplot(2,2,2), text(-0.5,1,0.5,'(b)','Fontsize',[14])
subplot(2,2,3), text(-0.5,1,0.5,'(c)','Fontsize',[14])
subplot(2,2,4), text(-0.5,1,0.5,'(d)','Fontsize',[14])
```

The SVD command pulls out the diagonal matrix along with the two unitary matrices **U** and **V**. The loop above processes the sum of the first, second and third modes. This gives the POD modes of interest. Figure 15.7 demonstrates the modal decomposition and the accuracy achieved with representing the surface with one, two and three POD modes. The first mode alone captures 92% of the surface while three modes produce a staggering 99.9% of the energy. This should be contrasted with Fourier methods, for instance, which would require potentially hundreds of modes to achieve such accuracy. *As stated previously, these are the best one, two and three mode approximations of the function $f(x,t)$ that can be achieved.*

To be more precise about the nature of the POD, consider the *energy* in each mode. This can be easily computed, or has already been computed, in the SVD, i.e. they are the singular values σ_j. In MATLAB, the energy in the one mode and three mode approximation is computed from

```
energy1=sig(1)/sum(sig)
energy3=sum(sig(1:3))/sum(sig)
```

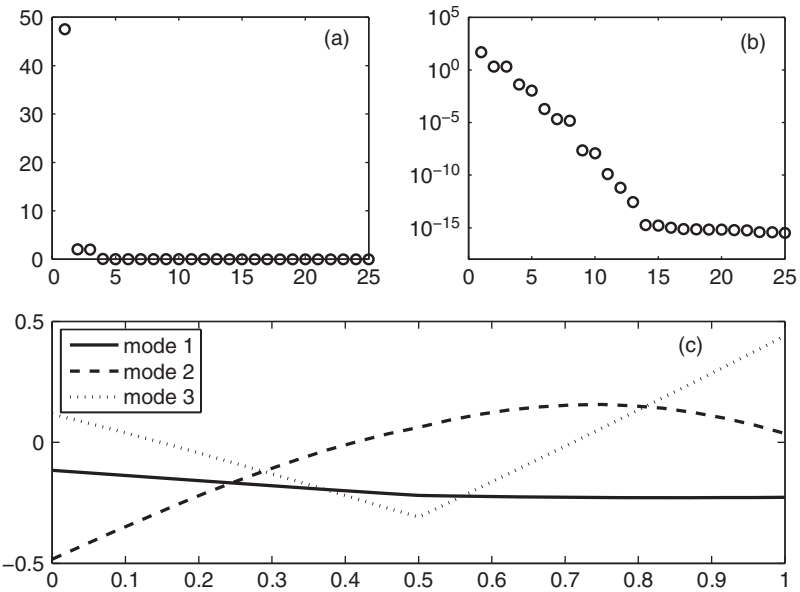

Figure 15.8: Singular values on a standard plot (a) and a log plot (b) showing the energy in each POD mode. For the surface considered in Fig. 15.7, the bulk of the energy is in the first POD mode. A three mode approximation produces 99.9% of the energy. The linear POD modes are illustrated in panel (c).

More generally, the entire *spectrum* of singular values can be plotted. Figure 15.8 shows the complete spectrum of singular values on a standard plot and a log plot. The clear dominance of a single mode is easily deduced. Additionally, the first three POD modes are illustrated: they are the first three columns of the matrix **U**. These constitute the orthonormal expansion basis of interest. The code for producing these plots is as follows:

```
sig=diag(s);
subplot(2,2,1), plot(sig,'ko','Linewidth',[1.5])
axis([0 25 0 50])
set(gca,'Fontsize',[13],'Xtick',[0 5 10 15 20 25])
text(20,40,'(a)','Fontsize',[13])
subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
axis([0 25 10^(-18) 10^(5)])
set(gca,'Fontsize',[13],'Ytick',[10^(-15) 10^(-10) 10^(-5) 10^0 10^5],...
    'Xtick',[0 5 10 15 20 25]);
text(20,10^0,'(b)','Fontsize',[13])

subplot(2,1,2)
plot(x,u(:,1),'k',x,u(:,2),'k--',x,u(:,3),'k:','Linewidth',[2])
set(gca,'Fontsize',[13])
legend('mode 1','mode 2','mode 3','Location','NorthWest')
text(0.8,0.35,'(c)','Fontsize',[13])
```

Note that the key part of the code is simply the calculation of the POD modes which are weighted, in practice, by the singular values and their evolution in time as given by the columns of **V**.

Example 2: Consider the following space–time function

$$f(x,t) = [1 - 0.5\cos 2t]\text{sech}\, x + [1 - 0.5\sin 2t]\text{sech}\, x \tanh x\,. \tag{15.5.6}$$

This represents an asymmetric, time-periodic breather of a localized solution. Such solutions arise in a myriad of contexts are often obtained via full numerical simulations of an underlying physical system. Let's once again apply the techniques of POD analysis to see what is involved in the dynamics of this system. A discretization of the system is first applied and made into a two-dimensional representation in time and space.

```
x=linspace(-10,10,100);
t=linspace(0,10,30);
[X,T]=meshgrid(x,t);
f=sech(X).*(1-0.5*cos(2*T))+(sech(X).*tanh(X)).*(1-0.5*sin(2*T));
subplot(2,2,1), waterfall(X,T,f), colormap([0 0 0])
```

Note that, unlike before, the matrix **f** is now a 30×100 matrix so that the x-values are along the columns and t-values are along the rows. This is done simply so that we can use the **waterfall** command in MATLAB.

The singular value decomposition of the matrix **f** produces the quantities of interest. In this case, since we want the x-value in the rows, the transpose of the matrix is considered in the SVD.

```
[u,s,v]=svd(f');
for j=1:3
  ff=u(:,1:j)*s(1:j,1:j)*v(:,1:j)';
  subplot(2,2,j+1)
  waterfall(X,T,ff'), colormap([0 0 0]), set(gca,'Zlim',[-1 2])
end
subplot(2,2,1), text(-19,5,-1,'(a)','Fontsize',[14])
subplot(2,2,2), text(-19,5,-1,'(b)','Fontsize',[14])
subplot(2,2,3), text(-19,5,-1,'(c)','Fontsize',[14])
subplot(2,2,4), text(-19,5,-1,'(d)','Fontsize',[14])
```

This produces the original function along with the first three modal approximations of the function. As is shown in Fig. 15.9, the two-mode and three-mode approximations appear to be identical. In fact, they are. The singular values of the SVD show that ≈83% of the energy is in

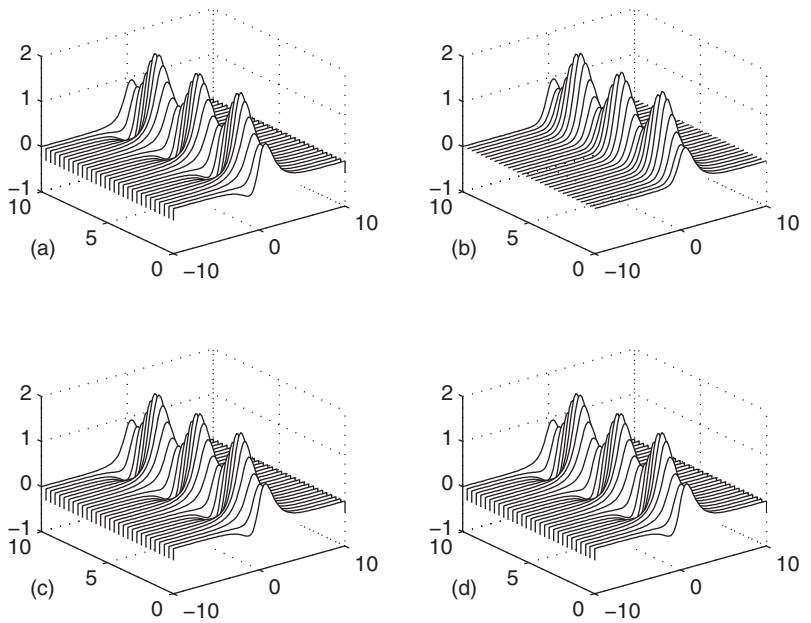

Figure 15.9: Representation of the space–time dynamics (a) by a series of low-rank (low dimensional) approximations with (b) one mode, (c) two modes and (d) three modes. The energy captured in the first mode is approximately 83% of the total energy. The first two modes together capture 100% of the surface energy, thus suggesting that the surface can be easily and accurately represented by a simple two-mode expansion.

the first mode of Fig. 15.9(a) while 100% of the energy is captured by a two mode approximation as shown in Fig. 15.9(b). Thus the third mode is completely unnecessary. The singular values are produced with the following lines of code.

```
figure(2)
sig=diag(s);
subplot(3,2,1), plot(sig,'ko','Linewidth',[1.5])
axis([0 25 0 50])
set(gca,'Fontsize',[13],'Xtick',[0 5 10 15 20 25])
text(20,40,'(a)','Fontsize',[13])
subplot(3,2,2), semilogy(sig,'ko','Linewidth',[1.5])
axis([0 25 10^(-18) 10^(5)])
set(gca,'Fontsize',[13],'Ytick',[10^(-15) 10^(-10) 10^(-5) 10^0 10^5],...
  'Xtick',[0 5 10 15 20 25]);
text(20,10^0,'(b)','Fontsize',[13])

energy1=sig(1)/sum(sig)
energy2=sum(sig(1:2))/sum(sig)
```

As before, we can also produce the first two modes, which are the first two columns of **U**, along with their time behavior, which are the first two columns of **V**.

```
subplot(3,1,2)  % spatial modes
plot(x,u(:,1),'k',x,u(:,2),'k--','Linewidth',[2])
set(gca,'Fontsize',[13])
legend('mode 1','mode 2','Location','NorthWest')
text(8,0.35,'(c)','Fontsize',[13])

subplot(3,1,3)  % time behavior
plot(t,v(:,1),'k',t,v(:,2),'k--','Linewidth',[2])
text(9,0.35,'(d)','Fontsize',[13])
```

Figure 15.10 shows the singular values along with the spatial and temporal behavior of the first two modes. It is not surprising that the SVD characterizes the total behavior of the system as an exactly (to numerical precision) two mode behavior. Recall that is exactly what we started with! Note, however, that our original two modes are different from the SVD modes. Indeed, the first SVD mode is asymmetric unlike our symmetric hyperbolic secant.

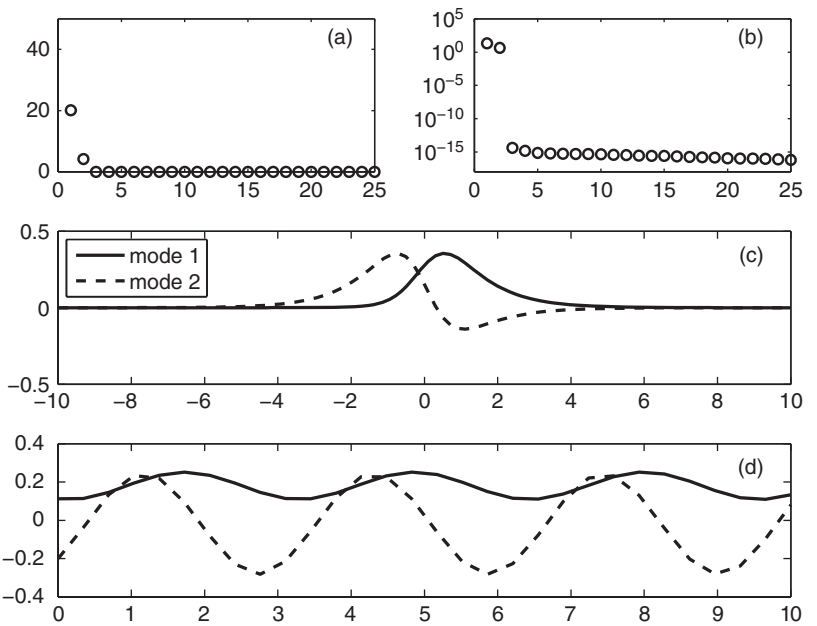

Figure 15.10: Singular values on a standard plot (a) and a log plot (b) showing the energy in each POD mode. For the space–time surface considered in Fig. 15.9, the bulk of the energy is in the first POD mode. A two mode approximation produces 100% of the energy. The linear POD modes are illustrated in panel (c) which their time evolution behavior in panel (d).

Summary and limits of SVD/PCA/POD

The methods of PCA and POD, which are essentially related to the idea of diagonalization of any matrix via the SVD, are exceptionally powerful tools and methods for the evaluation of data driven systems. Further, they provide the most precise method for performing low dimensional reductions of a given system. A number of key steps are involved in applying the method to experimental or computational data sets.

- Organize the data into a matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$ where m is the number of measurement types (or probes) and n is the number of measurements (or trials) taken from each probe.
- Subtract off the mean for each measurement type or row $\mathbf{x}_j$.
- Compute the SVD and singular values of the covariance matrix to determine the principal components.

If considering a fitting algorithm, then the POD method is the appropriate technique to consider and the SVD can be applied directly to the $\mathbf{A} \in \mathbb{C}^{m\times n}$ matrix. This then produces the singular values as well as the POD modes of interest.

Although extremely powerful, and seemingly magical in its ability to produce insightful quantification of a given problem, these SVD methods are not without limits. Some fundamental assumptions have been made in producing the results thus far, the most important being the assumption of *linearity*. Recently some efforts to extend the method using nonlinearity prior to the SVD have been explored [39, 40, 41]. A second assumption is that larger variances (singular values) represent more important dynamics. Although generally this is believed to be true, there are certainly cases where this assumption can be quite misleading. Additionally, in constructing the covariance matrix, it is assumed that the mean and variance are sufficient statistics for capturing the underlying dynamics. It should be noted that only exponential (Gaussian) probability distributions are completely characterized by the first two moments of the data. This will be considered further in the next section on independent component analysis. Finally, it is often the case that PCA may not produce the optimal results. For instance, consider a point on a propeller blade. Viewed from the PCA framework, two orthogonal components are produced to describe the circular motion. However, this is really a one-dimensional system that is solely determined by the phase variable. The PCA misses this fact unless use is made of the information already known, i.e. that a polar coordinate system with fixed radius is the appropriate context to frame the problem. Thus blindly applying the technique can also lead to nonoptimal results that miss obvious facts.

15.6 Robust PCA

The singular value decomposition and its various guises, most notably principal component analysis, are perhaps the most widely used statistical tools for generating dimensionality reduction of data. As has been demonstrated and discussed, the SVD produces characteristic features (principal components) that are determined by the covariance matrix of the data. Fundamental in

producing the SVD/PCA/POD modes is the L^2 norm for data fitting. Although the L^2 norm is highly appealing due to its centrality (and physical interpretation) in most applications, it does have potential flaws that can severely limit its applicability. Specifically, if corrupt data or large noise fluctuations are present in the data matrix, the higher dimensional least-square fitting performed by the SVD algorithm squares this pernicious error, leading to significant deformation of the singular values and PCA modes describing the data. Although many physical systems and/or computations are relatively free of data corruption, many modern applications in image processing, PIV fluid measurements, web data analysis, and bioinformatics, for instance, are rife with arbitrary corruption of the measured data, thus ensuring that standard application of the SVD will be of limited use.

Ideally, in performing dimensionality reduction one should not allow the corrupt or large noise fluctuations to so grossly influence one's results. In order to limit the impact of such *data outliers*, a different measure, or norm, can be envisioned in measuring the best data fit (SVD/PCA/POD modes). One measure that has recently produced great success in this arena is the L^1 norm [42] which no longer squares the distance of the data to the best-fit mode. Indeed, the L^1 norm has been demonstrated to promote sparsity and is the underlying theoretical construct in *compressive sensing* or *sparse sensing* algorithms (see Section 17.3 for further details). Here, the L^1 norm is simply used as an alternative measure (norm) for performing the equivalent of the least-square fit to the data. As a result, a more robust method is achieved of computing PCA components, i.e. the so-called *robust PCA* method.

Matrix decomposition: Low-rank plus sparse

To illustrate the entanglement of low-rank data with corrupt (sparse) measurement/matrix error, we can construct a new matrix which is composed of these two elements together

$$\mathbf{M} = \mathbf{L} + \mathbf{S} \tag{15.6.1}$$

where $\mathbf{M}$ is the data matrix of interest that is composed of a low-rank structure $\mathbf{L}$ along with some sparse data $\mathbf{S}$. Our objective is the following: given observations $\mathbf{M}$, can the low-rank matrix $\mathbf{L}$ and sparse matrix $\mathbf{S}$ be recovered? Adding to the difficulty of this task is the following: we do not know the rank of the matrix $\mathbf{L}$, nor do we know how many nonzero (sparse) elements of the matrix $\mathbf{S}$ exist.

Remarkably, Candés and co-workers [42] have recently proved that this can indeed be solved through the formulation of a convex optimization problem. At first sight, the separation problem seems impossible to solve since the number of unknowns to infer for $\mathbf{L}$ and $\mathbf{S}$ is twice as many as the given measurements in $\mathbf{M}$. Furthermore, it seems even more daunting that we expect to reliably obtain the low-rank matrix $\mathbf{L}$ with errors in $\mathbf{S}$ of arbitrarily large magnitude. But not only can this problem be solved, it can be solved by tractable convex optimization [42].

Of course, in real applications, it is rare that the matrix decomposition is as ideal as (15.6.1). More generally, the decomposition is of the form

$$\mathbf{M} = \mathbf{L} + \mathbf{S} + \mathbf{N} \tag{15.6.2}$$

where the matrix **N** is a dense, small perturbation accounting for the fact that the low-rank component is only approximately low-rank and that small errors can be added to all the entries (in some sense, this model unifies the classical PCA and the robust PCA by combining both sparse gross errors and dense small noise). But even in this case, the convex optimization algorithm formulated by Candés and co-workers [42] seems to do a remarkable job at separating low-rank from sparse data. This can be used to great advantage when certain types of data are considered, namely those with corrupt data or large, sparse noisy perturbations.

The details of the method and its proof are beyond the scope of this book. However, we will utilize the algorithms developed in the robust PCA literature in order to separate data matrices into low-rank and sparse components. A number of MATLAB algorithms can be downloaded from the web for this purpose (http://perception.csl.illinois.edu/matrix-rank/sample_code.html). Indeed, a wide variety of techniques have been developed for specifically providing a framework for robust PCA, including the augmented Lagrange multiplier (ALM) method [43], accelerated proximal gradient method [44], dual method [44], singular value thresholding method [45] and the alternating direction method [46]. For our purposes, we will use the MATLAB program **inexact_alm_rpca** since it is extremely simple to use and exceedingly fast [43].

Example: As an example, consider the function from the previous section

$$f(x,t) = [1 - 0.5\cos 2t]\text{sech}\, x + [1 - 0.5\sin 2t]\text{sech}\, x\, \tanh\, x\,. \tag{15.6.3}$$

Previously, the SVD was applied to this function and a two-mode dominance was clearly demonstrated (see Figs. 15.9 and 15.10). To illustrate the problem to be considered, the above function is plotted along with a corrupted version in Fig. 15.11. In this example, the data were corrupted by the addition of sparse, but large, noise fluctuations to the ideal data. The following MATLAB code produces the ideal figure.

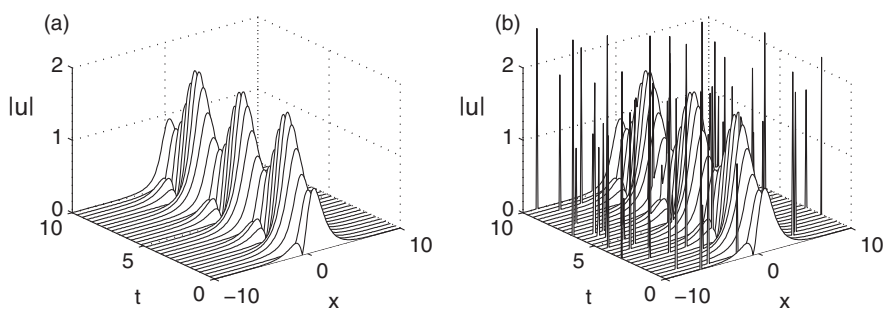

Figure 15.11: Representation of the space–time dynamics considered in Figs. 15.9 and 15.10. In (a), the ideal dynamics is demonstrated while in (b), a corrupted (noisy) image is assumed to be generated by the data collection process, for instance. The addition of noise brings into question the ability of the SVD to produce meaningful results. For this example, the low-rank matrix **L** has two modes of significance while the sparse matrix **S** has 60 nonzero entries.

```
n=200;
x=linspace(-10,10,n);
t=linspace(0,10,30);
[X,T]=meshgrid(x,t);
usol=sech(X).*(1-0.5*cos(2*T))+(sech(X).*tanh(X)).*(1-0.5*sin(2*T));
subplot(2,2,1), waterfall(x,t,abs(usol)); colormap([0 0 0]);
```

To add sparse noise, the **randintrlv** is used to produce noise spikes (60 spikes in total) in certain matrix/pixel locations, thus corrupting the data matrix.

```
sam=60;
Atest2=zeros(length(t),n);
Arand1=rand(length(t),n);
Arand2=rand(length(t),n);
r1 = randintrlv([1:length(t)*n],793);
r1k= r1(1:sam);
for j=1:sam
   Atest2(r1k(j))=-1;
end
Anoise=Atest2.*(Arand1+i*Arand2);
unoise=usol+2*Anoise;
subplot(2,2,2), waterfall(x,t,abs(unoise)); colormap([0 0 0]);
```

Figure 15.11 shows the ideal data along with the corrupt, or more physically relevant, data that might be collected in practice. The complex white noise fluctuations are, of course, the problematic part of the dimensionality reduction issue. Recall that without such noise fluctuations, the ideal data can be faithfully approximated with a low-rank, two-mode approximation.

The PCA reduction of the data matrices can now be compared by using the singular value decomposition. The following code produces the SVD decomposition of both data matrices

```
A1=usol; A2=unoise;
[U1,S1,V1]=svd(A1);
[U2,S2,V2]=svd(A2);
```

Both the singular values and the modal structures are represented in Figs. 15.12 and 15.13, respectively. Specifically, the top panel of Fig. 15.12 shows the two-mode dominance (top panel of Fig. 15.13) of the ideal data while the middle panels of these figures show the singular value

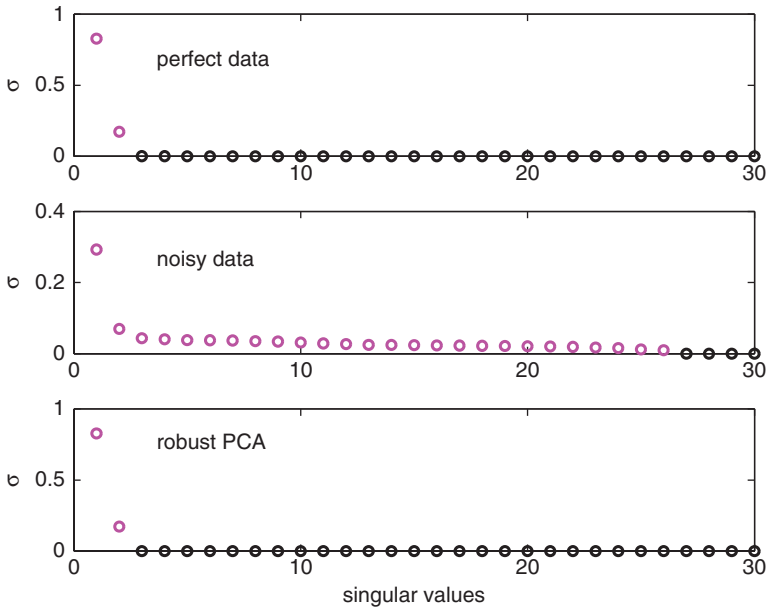

Figure 15.12: Singular values of the data matrices for the ideal data (top panel), the corrupted data (middle panel), and the low-rank data computed through robust PCA (bottom panel). The robust PCA construction produces almost a perfect match to the original, ideal data. The magenta dots represent the number of modes necessary to construct 99% of the original data. In this case, the ideal and robust PCA require two modes while the corrupt data require 26 modes.

distribution and the first four modes, respectively. Note that the addition of the corrupt data activates many more modes. In fact, it takes 26 modes to capture 99% of the energy of the data compared to two modes previously. The modes are also highly perturbed from their ideal states due to the sparse (corrupt) noise that was added.

To apply the robust PCA algorithm, the corrupt data matrix is decomposed into real and imaginary parts before applying the **inexact_alm_rpca** algorithm. This algorithm effectively separates the data into low-rank and sparse components as given by (15.6.1). The low-rank portion is kept in order to then acquire the PCA modes necessary for reconstruction.

```
ur=real(unoise);
ui=imag(unoise);

lambda=0.2;
[R1r,R2r]=inexact_alm_rpca(real(ur.'),lambda);
[R1i,R2i]=inexact_alm_rpca(real(ui.'),lambda);
```

Continued

```
R1=R1r+i*R1i;
R2=R2r+i*R2i;

[U3,S3,V3]=svd(R1.');

subplot(2,2,1), waterfall(x,t,abs(R1)'), colormap([0 0 0])
subplot(2,2,2), waterfall(x,t,abs(R2)'), colormap([0 0 0])
```

Note here that the real and imaginary portions are put back together at the end of the algorithm in order to SVD the low-rank portion. Further, the two matrices corresponding to the low-rank ($R1$) and sparse ($R2$) portions of the data matrix are plotted. The parameter *lambda* used in the **inexact_alm_rpca** algorithm can be tuned to best separate the sparse from low-rank matrices in (15.6.1). Here a value of 0.2 is used, which produces an almost ideal separation as is shown in Figs. 15.12 (bottom panel), 15.13 (bottom panel) and 15.14. Indeed, an almost perfect separation is achieved as advertised (guaranteed) by Candés and co-workers [42].

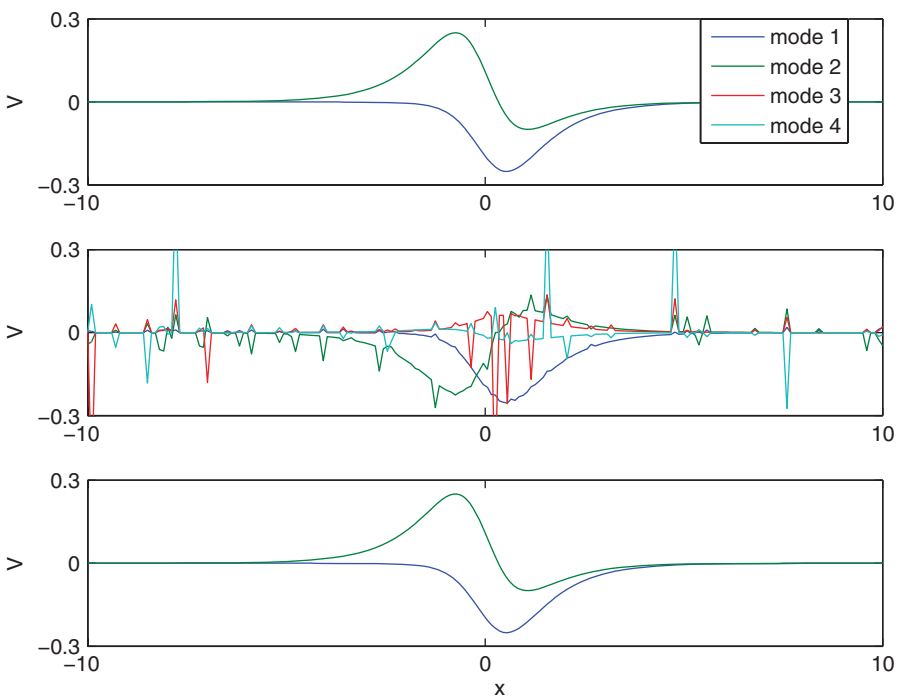

Figure 15.13: Dominant modes for the ideal data (top panel), the corrupted data (middle panel), and the low-rank data computed through robust PCA (bottom panel). The ideal and robust PCA produce the same dominant two modes while the corrupt data produce highly erratic modes.

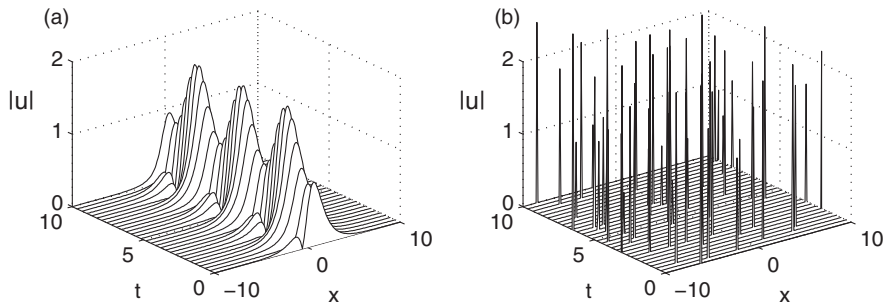

Figure 15.14: Separation of the original corrupt data matrix shown in Fig. 15.11(b) into a low-rank component (a) and a sparse component (b). The separation is almost perfect, with the low-rank matrix being dominated by two modes, i.e. they contain greater than 99% of the energy.

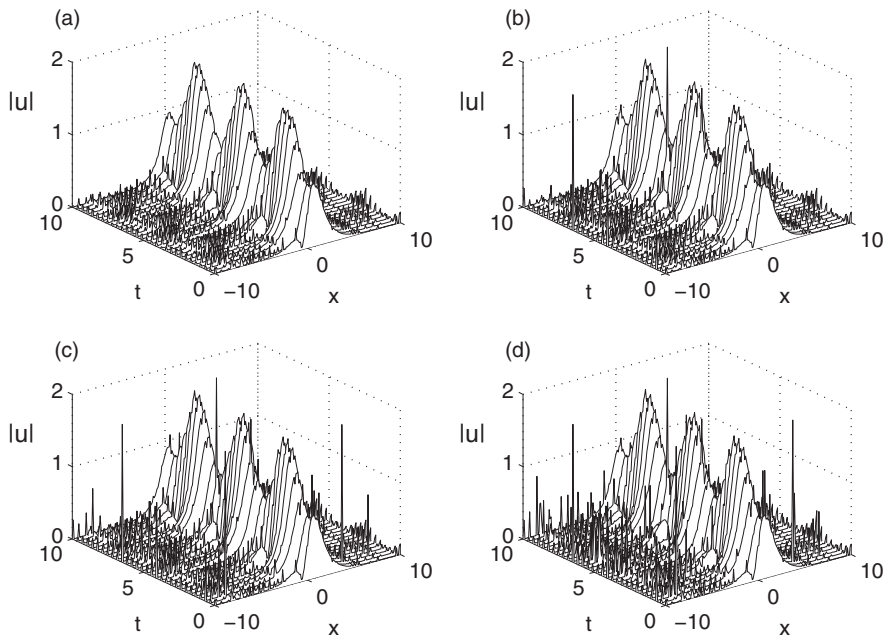

Figure 15.15: (a) Two-, (b) three-, (c) four- and (d) five-mode reconstruction of the matrix using the corrupted (sparse plus low-rank) data. Note that the addition of the higher modes adds sparse data spikes as is expected since they have not been filtered out of the data matrix.

To further investigate the robust PCA algorithm and its ability to extract the meaningful, low-rank matrix from the full matrix corrupted by sparse fluctuations, a series of low-rank approximations is made of the data matrices using the PCA modes generated by the SVD (see Figs. 15.15 and 15.16). The low-rank approximations show how effective the robust PCA algorithm is in separating out the low-rank from sparse components.

```
% low-rank approximation of corrupt data matrix
figure(1)
for jj=1:4
  for j=1:jj+1
    ff=U2(:,1:j)*S2(1:j,1:j)*V2(:,1:j).';
    subplot(2,2,jj), waterfall(x,t,abs(ff)), colormap([0 0 0])
  end
end

% approximation of low-rank matrix from robust PCA
figure(2)
for jj=1:4
  for j=1:jj+1
    ff=U3(:,1:j)*S3(1:j,1:j)*V3(:,1:j).';
    subplot(2,2,jj), waterfall(x,t,abs(ff)), colormap([0 0 0])
  end
end
```

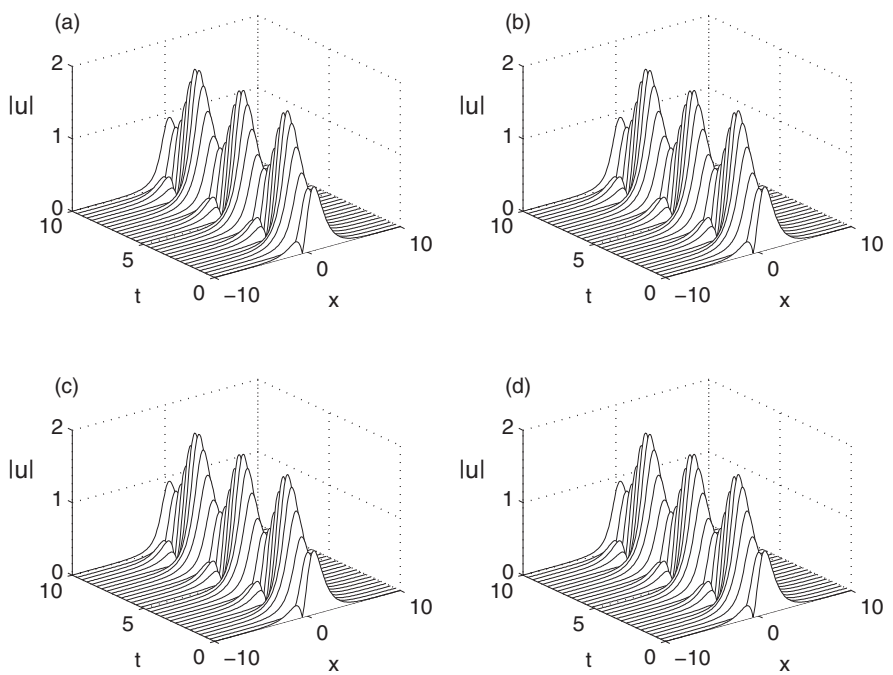

Figure 15.16: (a) Two-, (b) three-, (c) four- and (d) five-mode reconstruction of the matrix using the robust PCA algorithm for extracting the corrupted (sparse) components. Note that a two-mode expansion is sufficient to capture all the features of interest.

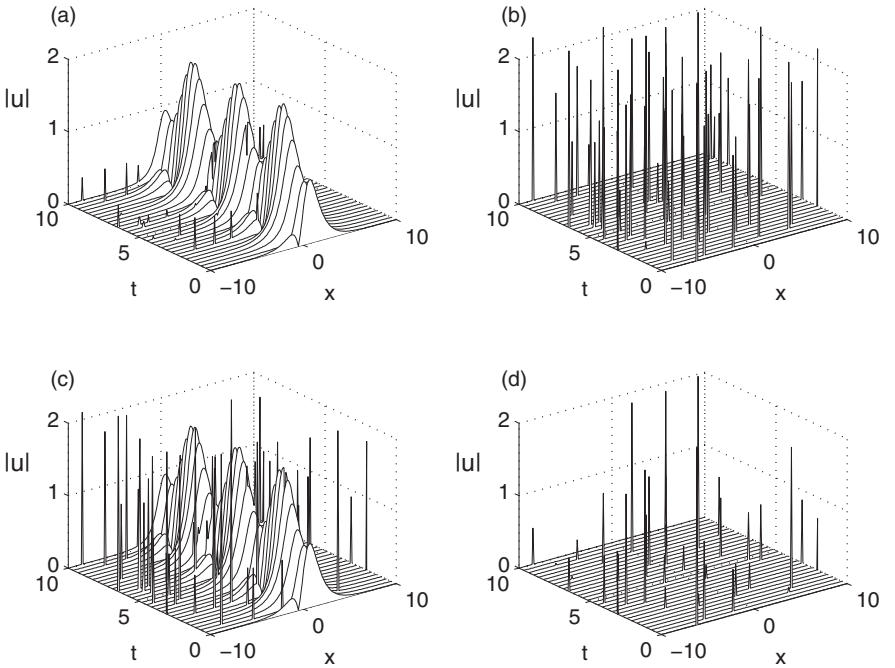

Figure 15.17: Separation of the original corrupt data matrix shown in Fig. 15.11(b) into a low-rank component (a) and (c) and a sparse component (b) and (d). The top panels are for *lambda* = 0.5 in the **inexact_alm_rpca** algorithm while the bottom panels are for *lambda* = 0.8. The separation deteriorates from the almost perfect separation illustrated previously with *lambda* = 0.2.

Finally, to end this discussion, the effects of the parameter *lambda* in the **inexact_alm_rpca** algorithm are illustrated. This parameter takes on values of zero to unity and can be tuned to achieve better or worse performance. The default, if *lambda* is not included, does a reasonable job generically in separating the low-rank from the sparse matrices. As *lambda* goes to zero, the computed low-rank matrix picks up the entire initial data matrix, i.e. the resulting sparse matrix is computed to be zero while the resulting sparse matrix contains the original data matrix. As *lambda* goes to unity, the opposite happens with the computed low-rank matrix containing nothing while the sparse matrix contains virtually the entire original matrix. Figure 15.17 depicts the results of the **inexact_alm_rpca** algorithm for *lambda* equal to 0.5 and 0.8. Note that as *lambda* is increased, the fidelity of the low-rank matrix is compromised and corrupted by the sparsity.

As a final comment, the robust PCA algorithm advocated for here is quite remarkable in its ability to separate sparse corruption from low-rank structure in a given data matrix. The application of robust PCA essentially acts as an advanced filter for cleaning up a data matrix. Indeed, one can potentially use this as a filter which is very unlike the time–frequency filtering schemes considered previously.