Ch. 15 - Linear Algebra and Singular Value Decomposition

We've encountered linear algebra on several occasions:

- The intrinsic case of solving systems of linear equations
- Curve fitting
- Most recently in linear programming optimization problems

Here we aim to:

- Generalize linear algebra methods (with more modern perspective)
- Develop tools that are especially useful in an era of data-driven analysis.

---

§15.1 Basics of the Singular Value Decomposition (SVD)

Most of important linear algebra methods that we have encountered can be both better understood and more compactly written as matrix decomposition/factorization:

- Gaussian elimination translated to LU factorization
    - $Ax = b \rightarrow LUx = b$ (where L and U are lower/upper triangular respectively)
    - Reduce solution of general linear system to a pair of triangular problems:
        - Solve $Ly = b$, then solve $Ux = y$
- Orthogonal basis (via Gram-Schmidt, Householder, etc.) led to QR decomposition
    - Orthogonal Q, "Rectangular" R

- Eigenvalue problems lead to eigendecomposition $A = S\Lambda S^{-1}$
    - Approach to obtaining the eigendecomposition:
        - Each eigenvalue/vector pair satisfies $Ax_i = \lambda_i x_i$
        - Collect the eigenvectors as columns of matrix S
        - Form Λ as diagonal matrix of corresponding eigenvalues
        - $\rightarrow AS = S\Lambda$
        - Right multiply by $S^{-1} \implies A = S\Lambda S^{-1}$

- Properties of eigendecomposition
    - Extremely useful for real, symmetric matrices
        - Arise frequently in applications (e.g. dynamics and vibrations)
        - Have real eigenvalues
        - Eigenvectors for distinct eigenvalues are orthogonal
        - All eigenvalues distinct $\implies$:
            - $\exists$ full set of orthogonal eigenvectors
            - Eigenvectors form orthogonal basis
            - Normalized eigenvectors as columns form an orthogonal matrix Q
            - $Q^{-1} = Q^T$ so the inverse does not require computation
            - $A = S\Lambda S^{-1} \rightarrow A = Q\Lambda Q^T$

- Limitations of eigendecomposition

- **Eigendecomposition does not exist for all matrices.**
- Fails for:
  - Rectangular A
  - S non-invertible (which is the general case when $A^T \neq A$)

> **Note on real vs. complex data:**

- Real data case:
  - Elements of A are real; notation is simple/familiar
  - Orthogonal matrices for which $Q^T = Q^{-1}$.
- Complex data case:
  - Replace "orthogonal matrix" with "unitary matrix"
  - Change "transpose" to "adjoint" (or "tranjugate" = "transpose + conjugate")

$$\left(Q^* = \overline{Q}^T = Q^{-1}\right)$$

Singular Value Decomposition (SVD) provides the generalization of eigendecomposition that works for all matrices.

Where does that come from?

Consider matrix multiplication, $Ax = y$, with $m \times n$ matrix A (m rows $\times$ n columns).

- Conformability requires the length of x to match up with n (# of columns)
- Length of output y matches up with m (the number of rows of A)
- Multiplication by $m \times n$ matrix A takes an input x with length n and produces output y with length m
- $A : \mathbb{R}^n \rightarrow \mathbb{R}^m$ (A maps n-vectors to m-vectors)

Previously considered square matrices:

- Matrix multiplication mapped vectors into the same space (or equivalent) space
- Think of multiplication by A causing a vector to stretch and rotate.

For general rectangular matrices:

- Doesn't make sense to think of the output vector as being a rotated/stretched version of input vector
  - Input and output vectors "live" in differents spaces of different dimension.
  - No good way to compare length/direction of vectors in different spaces.

How to proceed for rectangular matrices?

- Relate items of the dimension of each space that we can compare.
  - A geometric image of this concept is to consider our input in $\mathbb{R}^N$ as an n-dimensional hypersphere (e.g. the collection of all unit-length vectors in $\mathbb{R}^n$).
  - Performing the operation "multiply by A" on the hypersphere means multiply all of the unit vectors in $\mathbb{R}^n$ by A.
  - Matrix multiplication on each input vector produces a vector in $\mathbb{R}^m$ and we want to characterize the collection of output vectors.

Geometric "bottom line":

- Multiplication by $A_{m\times n}$ transforms a hypersphere in $\mathbb{R}^n$ into a hyperellipse in $\mathbb{R}^m$
- Characterized by principle semi-axis lengths $\sigma_1, \sigma_2, \ldots, \sigma_m$ associated with the principal directional unit vectors $u_1, u_2, \ldots, u_m$ that arose from multiplication of a set of vectors $v_1, v_2, \ldots, v_n \in \mathbb{R}^n$ by the matrix A.

> Note:
>
> - Geometric dimension of the hyperellipse will match up with $r = rank(A)$ which can be less than $m \implies$ some singular values can be zero.
> - But, no need to consider $\sigma < 0$. Why not?
> - We are considering an entire hypersphere as input, so we could take the diametrically opposed input vector to undo any sign changes that might arise.

The idea that a particular input unit vector gives rise to a particular principal semi-axis of the ouput hyperellipse can be written as:

$$Av_j = \sigma_j u_j, \quad 1 \leq j \leq n$$

where the v_j vectors provide an orthogonal basis for $\mathbb{R}^n$ and the u_j vectors associated with the principal directions of the hyperellipse provide an ortogonal basis for some sub-space of $\mathbb{R}^m$ of dimension $d = r \leq n$.

Consistent with our idea of considering the collections of inputs and outputs, we group these individual relations to obtain the set of relations:

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} \begin{bmatrix} v_1 & v_2 & \ldots & v_n \\ & & & \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \ldots & u_n \\ & & & \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}$$

that can be written in matrix notation as

$$AV = \hat{U}\hat{\Sigma}$$

Let's consider a typical case where A is a tall, skinny matrix (with more rows than columns, so $m > n$). The matrices in the equation above are as follows:

- A is the $m \times n$ matrix being analyzed.
- $\hat{\Sigma}$ is an $n \times n$ diagonal matrix with the number of positive entries corresponding to $rank(A)$ (so with positive diagonal entries if A is of full rank).
- $\hat{U}$ is a tall, skinny ($m \times n$) matrix with orthonormal columns.
- $\hat{V}$ is an $n \times n$ orthogonal/unitary matrix.

Because V is orthogonal, $V^{-1} = V^T$ and we can solve for A (and complete the first version of the decomposition) by right-muliplying by V^T to get the **Reduced (or "Economy") SVD Decomposition**:

$$A = \hat{U}\hat{\Sigma}V^T$$

The shapes of the relevant matrices are illustrated in Fig. 15.3 below from the text by Kutz. (Remember that we are focusing on real matrices, so you can read V^* as V^T and "unitary" as "orthogonal" until you

encounter complex matrices.)

What is seen more frequently in the literature is the "full" SVD decomposition where U is augmented with $m-n$ additional columns to become an $m \times m$ square unitary matrix and Σ is augmented with $m-n$ rows of zeros as depicted in Fig. 15.4 above (again from the text by Kutz).

Here is a description of the matrices in the full SVD:

- U is $m \times m$ orthogonal/unitary
- V is $n \times n$ orthogonal/unitary
- Σ is $m \times n$ diagonal
- Diagonal entries of Σ are non-negative and ordered from largest to smallest:
  - $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$ where $p = min(m,n)$

All of this sets up a theorem assuring the existence of the SVD:

Theorem: Every matrix $A \in \mathbb{C}^{m \times n}$ has a singular value decompostion $A = U\Sigma V^*$. The singular values $\{\sigma_j\}$ are uniquely determined and the singular vectors $\{u_j\}$ and $\{v_j\}$ are uniquely determined up to complex scale factors of unit magnitude.

Specifying the SVD

Now that we know the SVD applies to all matrices, let's consider how to compute it.

An initial approach involves the normal matrix that we obtain by multiplying by A^T on either the right or left:

$$\begin{aligned} A^TA &= \left(U\Sigma V^T\right)^T \left(U\Sigma V^T\right) \\ &= \left(U\Sigma V^T\right)^T \left(U\Sigma V^T\right) \\ &= V\Sigma U^T U\Sigma V^T \\ &= V\Sigma^2 V^T \end{aligned}$$

and

$$\begin{aligned} AA^T &= \left(U\Sigma V^T\right) \left(U\Sigma V^T\right)^T \\ &= \left(U\Sigma V^T\right) \left(U\Sigma V^T\right)^T \\ &= U\Sigma V^T V\Sigma U^T \\ &= U\Sigma^2 U^T \end{aligned}$$

Multiplying on the right by V and U gives

$$\begin{aligned} A^TAV &= V\Sigma^2 \\ AA^TU &= U\Sigma^2 \end{aligned}$$

Note that we have now recovered versions of the eigenproblem $MV = V\Lambda$ which describes how a square matrix M operates to stretch (and possibly reverse the direction of) eigenvectors v_j to produce $\lambda_j v_j$.

We can now conclude that:

- The left singular vectors $\{u_j\}$ of the SVD correspond to the eigenvectors of AA^T

- The right singular vectors $\{v_j\}$ of the SVD correspond to the eigenvectors of A^TA
- The singular values of $\{\sigma_j\}$ are the positive eigenvalues of AA^T (or A^TA)

Recall from our discussion of normal matrices in the context of least-squares fitting, actually doing the matrix multiplication to produce a normal matrix increases the condition number and has a detrimental effect on numerical accuracy.

> More appropriate methods for computing the SVD avoid dealing with the normal matrix (and its bad effect on condition number).

An alternative view of the SVD

Pieces of the SVD (*à la* Strang) ("Linear Algebra and Learning from Data," Wellesley, 2019)

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \sigma_3 u_3 v_3^T + \ldots + \sigma_r u_r v_r^T$$

- Each term involves a scalar multiple (specifically a singular value) of a product of:
  - a basis vectors from $\mathbb{R}^m$ (u_i with dimensions $m \times 1$)
  - the transpose of a basis vector from $\mathbb{R}^n$ (v_j^T with dimensions $1 \times n$)
  - produces an $m \times n$ matrix (i.e., the same dimensions as the original matrix A).
  - specifies a **rank 1 matrix** because it is the outer product of 1 vector from each of the relevant spaces.

The "pieces of the SVD" equation expresses an important idea:

- Any matrix A with $rank(A) = r$ decomposes into a sum of r rank 1 matrices.
- Significance of any term depends on its singular value σ_j (u_j, v_j are unit vectors).
- Mutual orthogonality of the $u_j, v_j \implies \exists$ subspaces from truncation of u_j and v_j:
  - $R_1 = span\left(u_1\right) \cdot span\left(v_1^T\right)$
  - $R_2 = span\left(u_1, u_2\right) \cdot span\left(v_1^T, v_2^T\right)$
  - $R_2 = span\left(u_1, u_2\right) \cdot span\left(v_1^T, v_2^T\right)$
  - $R_r = span\left(u_1, u_2, \ldots, u_r\right) \cdot span\left(v_1^T, v_2^T, \ldots, v_r^T\right)$
  - $R_1 \subseteq R_2 \subseteq \ldots \subseteq R_r$

- Implications for approximation based on SVD
  - The best approximation of A in R_1 is $\sigma_1 u_1 v_1^T$
  - The best approximation of A in R_2 is $\sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T$.
  - The best approximation of A in R_d is $\sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_d u_d v_d^T$
  - The best approximation of A in R_r is

$$\sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \ldots + \sigma_r u_r v_r^T$$

  - These are results of the Eckert-Young Theorem that dates back to 1936!

The big idea (or should I say "big data idea") goes as follows:

- "Gist" of a large matrix can be captured by a small number of well chosen vectors.
- Choose subspace with reasonable number of basis vectors, inspect singular values.
- Once σ_j is small, approximation of A improves only slightly in larger subspace.

> For those interested in data-driven analysis, numerous examples and applications are presented in Chapters 15-23 and 26; and Strang's book presents ample material on the mathematical, geometrical, and computational aspects of the SVD and related concepts. Also, you can check out the awesome videos on Prof. Brunton's YouTube channel...

Computing the SVD

As the SVD has become frequently used, a LOT of effort has gone into developing accurate and efficient methods for actually computing SVDs. See "Computation of the Singular Value Decomposition" by Cline and Dhillon (Ch. 58 of "Handbook of Linear Algebra," 2nd Edition by Leslie Hogben) for details.

SVD computation is sufficiently involved that it is advisable to use well-tested (and well-maintained) library functions to perform the SVD computation. In Python, the SVD is included in the linear algebra and scientific computation libraries:

- `numpy.linalg.svd` computes the SVD.
- `scipy.linalg.svd` also computes the SVD.
- `scipy.linalg.svdvals` computes only the singular values and not full factorization.

An approach to computing the SVD for which you already understand the basics:

- Based on an idea due to Lanczos circa 1961.
- Instead of considering just the $m \times n$ matrix A or its transpose/adjoint, group them together a follows:

$$\begin{bmatrix} 0_{m\times m} & A \\ A^T & 0_{n\times n} \end{bmatrix}$$

- Eigenvalues turn out to be the singular values of A (repreated with negative sign).
- Eigenvectors include right singular vectors as first m entries and left singular vectors as final n entries. (Some renormalization will be required.)

Small example of executing the Lanczos approach:

- Form the matrix M
- Compute the eigenvalues and eigenvectors
- Identify the singular values and singular vectors.
- Show that SVD provides information to reconstruct A as sum of rank 1 matrices.

```
import numpy as np
np.set_printoptions(precision=2, suppress=True)
# define a sample matrix A and form the M matrix as suggested by Lanczos
A = 1.*np.array([[1,2],[3,4],[5,6]])
m,n = A.shape
M = np.block([[np.zeros([m,m]), A],[A.T, np.zeros([n,n])]])
A, A.T,M
```

$$\begin{aligned}(&\text{array}([[1., 2.],\\ &\qquad[3., 4.],\\ &\qquad[5., 6.]]),\\ &\text{array}([[1., 3., 5.],\\ &\qquad[2., 4., 6.]]),\\ &\text{array}([[0., 0., 0., 1., 2.],\\ &\qquad[0., 0., 0., 3., 4.],\\ &\qquad[0., 0., 0., 5., 6.],\\ &\qquad[1., 3., 5., 0., 0.],\\ &\qquad[2., 4., 6., 0., 0.]]))\end{aligned}$$

```
# compute the eigenvalues and eigenvectors of M using a python library function
# Note that you already know a way to do this based on what you did in your homeworks
# Find leading eigenvalue/vector by matrix iteration
# Construct projection matrix P to collapse along the direction of the lead eigenvect
# Perfrom iteration with the matrix PA to find the second leading eigenvalue/vector
vals, vecs = np.linalg.eig(M)
vals, vecs
```

$$\begin{aligned}(&\text{array}([9.53, -9.53, 0.51, -0.51, -0.]),\\ &\text{array}([[0.16, 0.16, -0.62, 0.62, 0.41],\\ &\qquad[0.37, 0.37, -0.17, 0.17, -0.82],\\ &\qquad[0.58, 0.58, 0.28, -0.28, 0.41],\\ &\qquad[0.44, -0.44, 0.56, 0.56, 0.],\\ &\qquad[0.56, -0.56, -0.44, -0.44, -0.]]))\end{aligned}$$

```
s0 = vals[0] # Assign leading (largest positive) eigenvalue as lead singular value s0
evec0 = vecs.T[0] # Assign associated eigenvector as evec0
s0,evec0
```

$$(9.525518091565104, \text{array}([0.16, 0.37, 0.58, 0.44, 0.56]))$$

```
# Check that evec0 is the eigenvector of M with eigenvalue s0
evec0, M@evec0/s0
```

(array([0.16, 0.37, 0.58, 0.44, 0.56]), array([0.16, 0.37, 0.58, 0.44, 0.56]))

```
# Identify right singular vector as last n entries in the eigenvector
v0 = evec0[-2:]
v0 = v0/np.linalg.norm(v0) #normalize the singular vector
v0
```

array([0.62, 0.78])

```
# Identify left singular vector as first m entries in the eigenvector
u0 = evec0[0:3]
u0 = u0/np.linalg.norm(u0) #normalize the singular vector
u0
```

array([0.23, 0.52, 0.82])

```
# Compute first approximation of A based on 1 left/right singular vector and display
s0*np.outer(u0,v0), A
```

$$\begin{aligned}(&\text{array}([[1.36, 1.72],\\ &\qquad[3.1, 3.92],\\ &\qquad[4.84, 6.13]]),\\ &\text{array}([[1., 2.],\\ &\qquad[3., 4.],\\ &\qquad[5., 6.]]))\end{aligned}$$

```
#Identify the next singular value s1 (as next largest positive eigenvector of M)
s1 = vals[2]
evec1 = vecs.T[2] #assign associated eigenvector to evec1
s1,evec1
```

(0.5143005806586438, array([-0.62, -0.17, 0.28, 0.56, -0.44]))

```
# Assign v1, u1 as first 3 and last 2 entries of evec1 and normalize
v1 = evec1[-2:]
v1 = v1/np.linalg.norm(v1)
u1 = evec1[0:3]
u1 = u1/np.linalg.norm(u1)
s1, u1, v1
```

(0.5143005806586438, array([-0.88, -0.24, 0.4]), array([0.78, -0.62]))

```
# Compute best approximation of A based on 2 left/right singular vectors
s0*np.outer(u0,v0) + s1*np.outer(u1,v1)
```

$$\begin{aligned}(&\text{array}([[1., 2.],\\ &\qquad[3., 4.],\\ &\qquad[5., 6.]])\end{aligned}$$

- This sum of 2 rank 1 matrices reconstructs A (to within roundoff error)

- Consistent with the fact that $rank(A) = 2$.
- **Power of the SVD:** Matrices with large rank can often be well-approximated by a surprisingly small number of rank 1 terms based on singular values/vectors.
- Read on in $\text{Part } III$ of Kutz for more details...