CS 236756 - Technion - Intro to Machine Learning

Tal Daniel

Tutorial 03 - Linear Algebra & SVD

- Inspired by slides by Elad Osherov and slides from [MMDS (http://www.mmds.org/)](http://www.mmds.org/)

Agenda

- Linear Algebra Refresher
- Eigen Values and Vectors Decomposition
- Singular Value Decomposition

Useful Resource

[The Matrix Cookbook (http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf)](http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf)

In [1]:
```
# imports for the tutorial
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib notebook
```

Linear Algebra Refresher

Vectors

- Geometric object that has both a magnitude and direction
  - $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = (x_1, x_2, \dots, x_n)^T \in \mathcal{R}^n$
- Magnitude of a vector: $||x|| = \sqrt{x^T x} = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$
- **Cardinality** of a vector - the number of non zero elements

In [3]:

```
# let's see some vectors
v = np.random.randint(low=-20, high=20, size=(6, 1))
print("v:")
print(v)
print("v^T:")
print(v.T)
print("magnitude of v:")
print(np.sqrt(np.sum(np.square(v))))
print("cardinality- non zero elements:")
print(np.sum(v != 0))
```

```
v:
[[ -6]
 [ 10]
 [  0]
 [ 15]
 [-18]
 [-12]]
v^T:
[[ -6  10   0  15 -18 -12]]
magnitude of v:
28.792360097775937
cardinality- non zero elements:
5
```

Inner Product Space

- A mapping $\langle \cdot, \cdot \rangle : V \times V \rightarrow F$ that satisfies:
  - Conjucate Symetry: $\langle x, y \rangle = \overline{\langle y, x \rangle}$
  - Liniearity in the First Argument:
    - $\langle a \cdot x, y \rangle = a \cdot \langle x, y \rangle$
    - $\langle x + z, y \rangle = \langle x, y \rangle + \langle z, y \rangle$
  - Positive-definiteness:
    - $\langle x, x \rangle \geq 0$
    - $\langle x, x \rangle = 0 \rightarrow x = 0$
- Common Inner Products:
  - Real Vector: $\langle x, y \rangle = x^T y$
  - Real Matrix: $\langle A, B \rangle = trace(AB^T)$
  - Random Variables: $\langle x, y \rangle = \mathbb{E}[x \cdot y]$
- Properties of **Dot Product**:
  - Distributiveness:
    - $(a + b) \cdot c = a \cdot c + b \cdot c$
    - $a \cdot (b + c) = a \cdot b + a \cdot c$
  - Linearity: $(\lambda a) \cdot b = a \cdot (\lambda b) = \lambda(a \cdot b)$
  - Symetry: $a \cdot b = b \cdot a$
  - Non-Negativity: $\forall a \neq 0, a \cdot a > 0, a \cdot a = 0 \iff a = 0$

In [3]:
```python
# Let's see some dot products
a = np.ones((5,1))
b = np.random.randint(low=-10, high=10, size=(5,1))
print("a:")
print(a)
print("b:")
print(b)
print("a.T.dot(b)=")
print(a.T.dot(b))
print("the same as a.T @ b:")
print(a.T @ b)
print("a + 0.5=")
print(a + 0.5)
print("(a + 2 * a).T @ b")
print((a + 2 * a).T @ b)
print("the same as a.T @ b + (2 * a).T @ b")
print(a.T @ b + (2 * a).T @ b)
```

```
a:
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
b:
[[ -6]
 [  5]
 [ -8]
 [ -5]
 [-10]]
a.T.dot(b)=
[[-24.]]
the same as a.T @ b:
[[-24.]]
a + 0.5=
[[1.5]
 [1.5]
 [1.5]
 [1.5]
 [1.5]]
(a + 2 * a).T @ b
[[-72.]]
the same as a.T @ b + (2 * a).T @ b
[[-72.]]
```

Outer Product

- Let:
  - $a = (a_1, a_2, \ldots, a_n)^T$
  - $b = (b_1, b_2, \ldots, b_n)^T$
- The outer product ab^T:

$$ab^T = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} [b_1, b_2, \ldots, b_n] = \begin{pmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_n \\ a_2b_1 & a_2b_2 & \cdots & a_2b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_nb_1 & a_nb_2 & \cdots & a_nb_n \end{pmatrix}$$

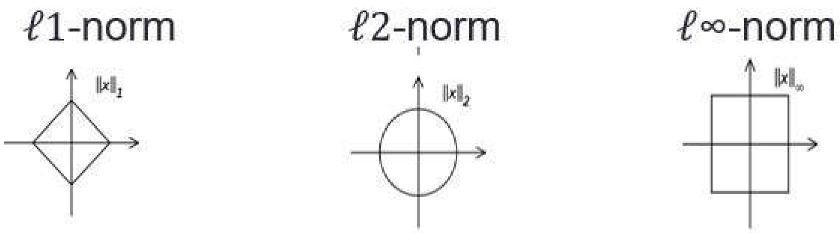
In [4]:

```
# outer product
a = np.random.random(size=(5,1))
print("a:")
print(a)
b = np.random.random(size=(5,1))
print("b:")
print(b)
ab_t = a @ b.T
print("outer product: a @ b.T = ")
print(ab_t)
```

```
a:
[[0.90266288]
 [0.94672242]
 [0.6097391 ]
 [0.82390305]
 [0.887672  ]]
b:
[[0.17321164]
 [0.658294  ]
 [0.28232724]
 [0.52144122]
 [0.47426751]]
outer product: a @ b.T =
[[0.15635172 0.59421756 0.25484632 0.47068563 0.42810367]
 [0.16398334 0.62322169 0.26728553 0.49366009 0.44899968]
 [0.10561391 0.40138759 0.17214596 0.3179431  0.28917944]
 [0.1427096  0.54237044 0.23261027 0.42961701 0.39075045]
 [0.15375512 0.58434916 0.25061399 0.46286877 0.42099399]]
```

Vector Norms

- A norm on a vector sapce Ω is a function $f : \Omega \rightarrow \mathcal{R}$ with the following properties:
  - Positive Scalability: $f(ax) = |a|f(x)$
  - Triangle Inequality: $f(x + y) \leq f(x) + f(y)$
  - If $f(x) = 0 \rightarrow x = 0$
- l_1 norm: $||x||_1 = \sum_{i=1}^{n} |x_i|$
- l_2 norm: $||x||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$
  - For **Vectors**: $||x||_2^2 = x^T x$
  - l_2-distance: $||x - y||_2^2 = (x - y)^T(x - y) = ||x||_2^2 - 2x^T y + ||y||_2^2$
- l_p norm: $||x||_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$
- l_∞ norm: $||x||_\infty = \max(|x_1|, |x_2|, \ldots, |x_n|)$



In [5]:
```
# norms and distance
a = np.random.random(size=(5,1))
print("a:")
print(a)
print("l-1 norm: ")
print(np.sum(abs(a)))
print("l-2 norm: ")
print(np.sqrt(np.sum(np.square(a))))
print("l-infinity norm:")
print(np.max(abs(a)))
b = np.random.random(size=(5,1))
print("b:")
print(b)
print("l-2 distance between a and b:")
print(np.sqrt((a - b).T @ (a - b)))
```

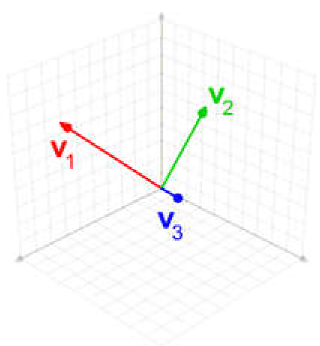
```
a:
[[0.21816978]
 [0.0166349 ]
 [0.77464535]
 [0.57721965]
 [0.1185309 ]]
l-1 norm:
1.7052005805479906
l-2 norm:
0.997588236132957
l-infinity norm:
0.7746453522517625
b:
[[0.08994736]
 [0.29621743]
 [0.17487165]
 [0.40283392]
 [0.15783109]]
l-2 distance between a and b:
[[0.69734551]]
```

Linear Dependency

- Given a set of vectors $X = \{x_1, x_2, \dots, x_n\}$, a **linear combination** of vectors is written as:
  $$ax = a_1x_1 + a_2x_2 + \dots + a_nx_n$$
- $x_i \in X$ is **linearly dependent** if it can be written as linear combination of $X \setminus \{x_i\}$

Basis

- A basis is a **linearly independent** set of vectors that spans the "whole sapce"
- Every vector in the space can be written as a linear combination of vectors in the basis
  - For example, **the standard basis (unit vectors)**: $\{e_i \in \mathcal{R}^n | e_i = (0, 0, \dots, 0, 1, 0, \dots, 0)^T\}$
    - $x^T = (3, 2, 5)^T = 3(1, 0, 0)^T + 2(0, 1, 0)^T + 5(0, 0, 1)^T = 3e_1^T + 2e_2^T + 5e_3^T$
- **Projection** of a vector: $x \cdot e_i = x^T e_i = e_i^T x$
- The basis vectors suffice:
  - Orthogonal - $e_i^T e_j = 0$
  - Normalized - $e_i^T e_i = 1$
  - Orthogonal + Normalized = Orthonormal
  - If A is **orthogonal** then:
    - A is a square matrix
    - The columns of A are **orthonormal** vectors
    - $A^T A = AA^T = I \rightarrow A^T = A^{-1}$
- **Change of Basis** - suppose that we have a basis not necessarily orthonormal $B = \{b_1, b_2, \dots, b_n\}, b_i \in \mathcal{R}^m$
  - Vector in the **new** basis is represented with a **matrix-vector** multiplication
  - The Identity matrix I maps a vector to itself
  - Basis change can be decomposed to: **rotation** matrix and **scale** matrix
  - Using an **orthonormal** basis means only a **rotation** around the origin
  - **Gram-Schmidt Orthonormaliztion Process**: Link (https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process)



By Lucas V. Barbosa (//commons.wikimedia.org/wiki/User:Kieff) - Own work, Public Domain, Link (https://commons.wikimedia.org/w/index.php?curid=24396471)

In [6]:

```
# Gram-Schmidt Algorithm
def gram_schmidt(V):
    """
    Implements Gram-Schmidt Orthonormaliztion Process.
    Parameters:
        V - matrix such that each column is a vector in the original basis
    Returns:
        U - matrix with orthonormal vectors as columns
    """
    n, k = np.array(V, dtype=np.float).shape  # get dimensions
    # initialize U matrix
    U = np.zeros_like(V, dtype=np.float)
    U[:,0] = V[:,0] / np.sqrt(V[:,0].T @ V[:,0])
    for i in range(1, k):
        U[:,i] = V[:,i]
        for j in range(i - 1):
            U[:,i] = U[:,i] - ((U[:,i].T @ U[:,j]) / (U[:,j].T @ U[:,j])) * U[:, j]
        # normalize
        U[:,i] = U[:,i] / np.sqrt(U[:,i].T @ U[:,i])
    return U

v1 = [3.0, 1.0]
v2 = [2.0, 2.0]
v = np.stack((v1, v2), axis=1)
print("V:")
print(v)
U = gram_schmidt(v)
print("U:")
print(U)
```

```
V:
[[3. 2.]
 [1. 2.]]
U:
[[0.9486833  0.70710678]
 [0.31622777 0.70710678]]
```

Matrix Operations

- Addition
  - Commutative: $A + B = B + A$
  - Associative: $(A + B) + C = A + (B + C)$
- Multiplication - **PAY ATTENTION TO DIMENSTIONS**
  - Associative: $A(BC) = (AB)C$
  - Distributive: $A(B + C) = AB + AC$
  - Non-comutative (**!**): $AB \neq BA$
- Transpose
  - $(A^T)_{ij}$
  - $(A^T)^T = A$
  - $(AB)^T = B^T A^T$
- Inverse - **MAKE SURE CONDITIONS APPLY**
  - **Positive Semi-definite (PSD)** - Matrix M is called *PSD* if for every non-zero column vector z, the scalar $z^T M z \geq 0$
  - **Every positive definite matrix is invertible** and its inverse is also positive definite
  - $(A^{-1})^{-1} = A$
  - $(AB)^{-1} = B^{-1} A^{-1}$
  - $(A^T)^{-1} = A^{-T}$
  - Inverse of 2x2 matrix: check tutorial 1

```
In [7]: # inverse
A = np.random.rand(5, 5)
print("A:")
print(A)
print("inverse of A:")
print(np.linalg.inv(A))
```

```
A:
[[0.98463396 0.45447991 0.49365401 0.24064992 0.54808428]
 [0.42278911 0.83835018 0.39599129 0.73846631 0.63763384]
 [0.35294769 0.00152597 0.9388472  0.31903281 0.2760794 ]
 [0.82842935 0.64966235 0.40236603 0.47731957 0.12382832]
 [0.93792802 0.43518515 0.46486903 0.2144443  0.12089541]]
inverse of A:
[[  3.53324514  -2.98005006   0.1124123    5.56530918  -6.2575821 ]
 [ -6.92616274   5.94500258  -1.5250271  -11.61645075  15.42540149]
 [ -4.03894987   2.89078871   0.56121474  -6.69081739   8.63550628]
 [  6.13826919  -5.40661561   1.42528243  14.0146561  -16.92165325]
 [  2.16302225   0.19417892  -0.06865877  -0.49267226  -1.89727223]]
```

Matrix Rank

- The rank of a matrix is the **maximal number of linearly independent** columns or rows of a matrix
- $A \in \mathcal{R}^{m \times n} \rightarrow rank(A) \leq \min(m, n)$
- $rank(A) = rank(A^T)$
- $rank(A^T A) = rank(A)$
- $rank(A + B) \leq rank(A) + rank(B)$
- $rank(AB) \leq \min(rank(A), rank(B))$
- A is **full rank** if $rank(A) = \min(m, n)$
- **Singular Matrix** - has dependent rows (and at least one zero eigen-value)

```
In [8]: A = np.random.randint(low=0, high=4, size=(5,5))
print("A:")
print(A)
print("rank(A):")
print(np.linalg.matrix_rank(A))
```

```
A:
[[1 0 1 3 1]
 [2 0 1 0 0]
 [1 2 3 0 3]
 [1 3 1 3 3]
 [1 3 3 3 1]]
rank(A):
5
```

Range & Nullspace

- **Range** (of a matrix) - the span of the columns of the matrix, denoted by the set:
$$\mathcal{R}(A) = \{y | y = Ax\}$$
- **Nullspace** (of a matrix) - the set of vectors that when multiplied by the matrix result in 0, given by the set:
$$\mathcal{N}(A) = \{x | Ax = 0\}$$

Determinant

Let $A = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix}$, a **square matrix**, then:

- $det(A) = |A| = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} = x_1 \begin{vmatrix} y_1 & z_2 \\ y_3 & z_3 \end{vmatrix} - x_2 \begin{vmatrix} y_1 & z_1 \\ y_3 & z_3 \end{vmatrix} + x_3 \begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix} = x_1(y_2z_3 - z_2y_3) - x_2(y_1z_3 - z_1y_3)$
$+ x_3(y_1z_2 - z_1y_2)$
- $det(A) = 0 \iff A$ is **singular** (at least one eigen-value is zero)
- If A is diagonal, then $det(A)$ is the prodcut of the diagonal elements (the eigen-values)
- $det(AB) = det(A)det(B)$
- $det(A^{-1}) = det(A)^{-1}$
- $det(\lambda A) = \lambda^n det(A)$

In [9]:
```
# determinant
A = np.random.randn(5,5)
print("A:")
print(A)
print("det(A):")
print(np.linalg.det(A))
```

```
A:
[[ 1.4792694  -0.56204891 -0.57136075 -2.1527347   0.79861811]
 [ 0.37725727 -0.11402182  0.55368031 -0.17979957 -0.3096385 ]
 [-0.78610946 -0.06434369  0.12422531  0.14632316  0.79623484]
 [ 0.69049244 -0.84221851 -0.60109636  0.40418209  0.88894044]
 [-0.24910082 -1.3595816   0.20798817 -1.35291227  1.21427935]]
det(A):
-2.3388728660269216
```

Solve Linear Equation Analytically

- Definitions:
  - $A \in \mathcal{R}^{n\times n}$
  - $x, b \in \mathcal{R}^{n\times 1}$
- The problem: find the solution of $Ax = b$
- Solution: if A is PSD (and thus invertible), then $x = A^{-1}b$
- What if $A \in \mathcal{R}^{m\times n}$, $x \in \mathcal{R}^{n\times 1}$, $b \in \mathcal{R}^{m\times 1}$?
  - A is no longer invertible!
- The problem redefined: find x that minimzes the distance from Ax to b, or more formally:

  $$\underset{x}{\operatorname{argmin}} ||Ax - b||_2^2$$

  (also called **least-squares** solution)

Reminder (Tutorial 01) - Vector & Matrix Derivatives

- $\nabla_x Ax = A^T$
- $\nabla_x x^T Ax = (A + A^T)x$
- $\frac{\partial}{\partial A} \ln |A| = A^{-T}$
- $\frac{\partial}{\partial A} Tr[AB] = B^T$

Exercise 1 - Least-Squares Solution

Given $A \in \mathcal{R}^{m\times n}$, $x \in \mathcal{R}^{n\times 1}$, $b \in \mathcal{R}^{m\times 1}$

Find x that minimzes the distance from Ax to b, or more formally:

$$\underset{x}{\operatorname{argmin}} ||Ax - b||_2^2$$

Solution 1

$$||Ax - b||_2^2 = (Ax - b)^T(Ax - b) = x^T A^T Ax - x^T A^T b - b^T Ax + b^T b$$
$$\frac{\partial ||Ax - b||_2^2}{\partial x} = 2A^T Ax - 2A^T b = 0 \rightarrow x = (A^T A)^{-1} A^T b$$

In [10]:

```
# Least Squares Solution
m = 5
n = 4
A = np.random.randint(low=-5, high=10, size=(m,n))
b = np.random.randint(low=-10, high=3, size=(m,1))
print("A:")
print(A)
print("b:")
print(b)
print("Least Squares solution for x:")
x = np.linalg.inv(A.T @ A) @ A.T @ b
print(x)
```

```
A:
[[-4  0 -1 -1]
 [ 0  3  6 -4]
 [-2  5  1 -4]
 [ 2  2  3  2]
 [ 6 -3 -4  2]]
b:
[[ 0]
 [ 1]
 [-1]
 [-9]
 [-4]]
Least Squares solution for x:
[[-0.59465397]
 [-1.76026098]
 [ 0.1814853 ]
 [-1.55940215]]
```

Solve Linear Equation Non-Analytically

Eigenvalues and Eigenvectors

- Definition: Matrix A with **Eigenvalue** $\lambda \in \mathbb{C}$ and **Eigenvector** $x \in \mathbb{C}^n$ if
$$Ax = \lambda x, x \neq 0$$
- Finding eigenvalues and eigenvectors
  - Find eigenvalues by finding the roots of the polynomial generated by:
  $$det(\lambda I - A) = |\lambda I - A| = 0$$
  - For each eigenvalue λ, find its corresponding eigenvector x by solving:
  $$Ax = \lambda x$$
- Example: $M = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \rightarrow |\lambda I - M| = \begin{vmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{vmatrix} = 3 - 4\lambda + \lambda^2 \rightarrow \lambda_{1,2} = 1, 3 \rightarrow x_{\lambda=1} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, x_{\lambda=3} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
- Eigenvalues Properties
  - $det(\Lambda) = |\Lambda| = \prod_{i=1}^{n} \lambda_i$
  - $rank(A) = \sum_{i=1}^{n} 1_{\lambda_i \neq 0}$
  - Eigenvalues of a **diagonal** matrix are the diagonal entries
  - A (square) matrix is said to be **diagonalizable** if it can be rewritten as: $A = X \Lambda X^{-1}$
- Eigenvalues of **Symmetric Matrices**:
  - Eigenvalues are **real**
  - Eigenvectors of **real symmetric** matrices are orthonormal

In [11]:
```python
# eigenvalues and eigenvectors
A = np.random.randint(low=-10, high=10, size=(5,5))
eig, vec = np.linalg.eig(A)
print("A:")
print(A)
print("eigenvalues:")
print(eig)
print("eigenvectors:")
print(vec)
```

```
A:
[[-4  3  7  5 -3]
 [-6  0 -7  1  5]
 [-3  6  8 -2  0]
 [-3  6  8  1  0]
 [ 6 -7 -3  4  1]]
eigenvalues:
[ 2.15100061+11.82247774j  2.15100061-11.82247774j
 -3.962019   +0.j          3.96421735 +0.j
  1.69580043 +0.j        ]
eigenvectors:
[[-0.12167344+0.41100288j -0.12167344-0.41100288j -0.70317398+0.j
   0.32865106+0.j         -0.33260964+0.j        ]
 [-0.20633163-0.48388658j -0.20633163+0.48388658j -0.67589276+0.j
   0.26261661+0.j         -0.71850371+0.j        ]
 [-0.30325931+0.18124115j -0.30325931-0.18124115j  0.17977307+0.j
   0.14085581+0.j          0.37207149+0.j        ]
 [-0.25200678+0.25451382j -0.25200678-0.25451382j  0.10230715+0.j
   0.57910496+0.j         -0.48379021+0.j        ]
 [ 0.53521396+0.j          0.53521396-0.j         -0.07700792+0.j
   0.68397228+0.j         -0.02516122+0.j        ]]
```

Eigen Decomposition

- **Eigen-decomposition** (also **spectral decomposition**) - factorization of a matrix into a canonical form, that is, the matrix is represented in terms of its **eigenvalues and eigenvectors**.
- **Only** diagonalizable matrices can be factorized
- Formally:
  - Denote Λ as a matrix with eigenvalues on the diagonal
  - Denote Q as a matrix where the columns are the eigenvectors
  - Let A be a square $n \times n$ matrix with N linearly **independent** columns. Then A can factorized as:
  $$A = Q\Lambda Q^{-1}$$

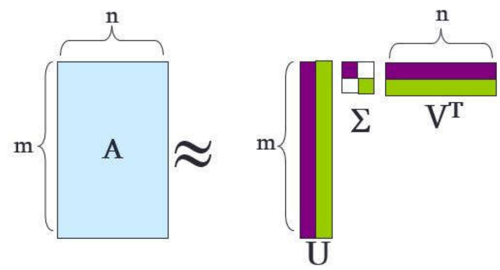
What If A Is Non-Square?

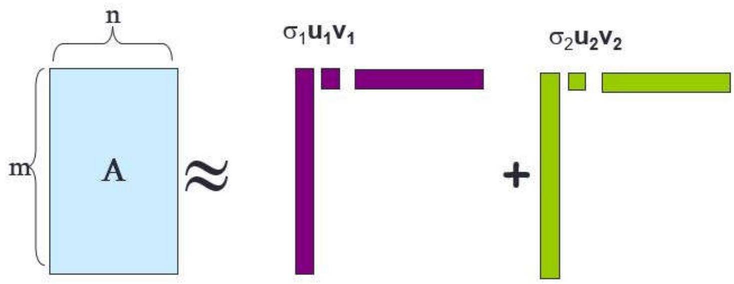
Singular Value Decomposition (SVD)

- In linear algebra, the singular-value decomposition (SVD) is a factorization of a real or complex matrix. It is the generalization of the eigendecomposition of a positive semidefinite normal matrix (for example, a symmetric matrix with positive eigenvalues) to any $m \times n$ matrix via an extension of the polar decomposition.
- Definition:
$$A_{[m\times n]} = U_{[m\times r]}\Sigma_{[r\times r]}(V_{[n\times r]})^T$$
- A - Input Data matrix
  - $m \times n$ matrix (e.g. m documents and n terms that can appear in each document)
- U - Left Singular vectors
  - $m \times r$ matrix (e.g. m documents and r concepts)
  - $U = eig(AA^T)$
- Σ - Singular values
  - $r \times r$ **diagonal** matrix (strength of each 'concept')
  - r represnts the **rank** of matrix A
  - $\Sigma = diag(\sqrt{eigenvalues(A^TA)})$
  - **Singular Values** definition: the singular values of a matrix $X \in \mathbb{R}^{M\times N}$ are the *square root* of the **eigenvalues** of the matrix $X^TX \in \mathbb{R}^{N\times N}$. If $X \in \mathbb{R}^{N\times N}$ already, then the singular values are the eigenvalues.
- V - Right Singular vectors
  - $n \times r$ matrix (e.g. n terms and r concepts)
  - $V = eig(A^TA)$
- Illustration:

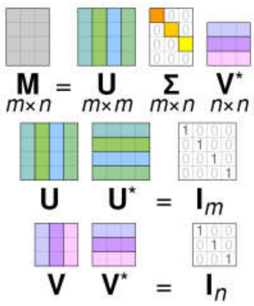
  First, we see the unit disc in blue together with the two canonical unit vectors. We then see the action of M, which distorts the disk to an ellipse. The SVD decomposes M into three simple transformations: an initial rotation V^*, a scaling Σ along the coordinate axes, and a final rotation U. The lengths σ_1 and σ_2 of the semi-axes of the ellipse are the singular values of M, namely $\Sigma_{1,1}$ and $\Sigma_{2,2}$.

  By Kieff (//commons.wikimedia.org/wiki/User:Kieff) - Own work, Public Domain, Link (https://commons.wikimedia.org/w/index.php?curid=11416486)
- Another way to look at SVD:
$$A \approx U\Sigma V^T = \sum_i \sigma_i u_i \circ v_i^T$$





- **SVD Properties**
  - It is **always** possible to decompose a **real** matrix A to $A = U\Sigma V^T$ where
    - U, Σ, V are **uniuqe**
    - U, V are column **orthonormal**
      - $U^TU = I, V^TV = I$
    - Σ is **diagonal**
      - Entries (the singular values) are positive and **sorted** in decreasing order ($\sigma_1 \geq \sigma_2 \geq ... \geq 0$)
  - Proof of uniqueness (http://www.mpi-inf.mpg.de/~bast/ir-seminar-ws04/lecture2.pdf!\[image.png\](attachment:image.png))



(Image from Wikipedia (https://en.wikipedia.org/wiki/Singular_value_decomposition))

SVD Example - Users-to-Movies

We are given a dataset of user's rating (1 to 5) for several movies of 3 genres (concepts) and we wish to use SVD to decompose to the following componnets:

- User-to-Concept - which genres the users prefer: U matrix
- Concepts - what is the strength of each genre in the dataset: Σ - strength of each concept (the singular values)
- Movie-to-Concept - for each movie, what genres are the most dominant: V matrix

In [12]:
```
# Load the dataset and create a pandas DataFrame
u_t_m = np.array([[1,1,1,0,0], [3,3,3,0,0], [4,4,4,0,0], [5,5,5,0,0], [0,2,0,4,4], [0,0,0,5,5], [0,1,0,2,2
]])
print("User-to-Movies matrix:")
# print(u_t_m)
u_t_m_df = pd.DataFrame(u_t_m, columns=['Matrix', 'Alien', 'Serenity', 'Casablanca', 'Amelie'],
                        index=['User 1', 'User 2','User 3', 'User 4', 'User 5', 'User 6', 'User 7'])
u_t_m_df
```

```
User-to-Movies matrix:
```

Out[12]:

	Matrix	Alien	Serenity	Casablanca	Amelie
User 1	1	1	1	0	0
User 2	3	3	3	0	0
User 3	4	4	4	0	0
User 4	5	5	5	0	0
User 5	0	2	0	4	4
User 6	0	0	0	5	5
User 7	0	1	0	2	2

In [13]:
```
# perform SVD for 3 concepts
u, s, vh = np.linalg.svd(u_t_m, full_matrices=False)
print("U of size", u[:,:3].shape, ":")
print(u[:,:3].astype(np.float16))
print("Singular values:")
print(s.astype(np.float16)[:3])
print("as a matrix:")
print(np.diag(s[:3]).astype(np.float16))
print("V of size", vh[:3,:].shape, ":")
print(vh[:3,:].astype(np.float16))

# reconstruct the user-to-movie matrix
A_aprox = u[:,:3] @ np.diag(s[:3]) @ vh[:3,:]
A_aprox_df = pd.DataFrame(A_aprox.astype(np.float16), columns=['Matrix', 'Alien', 'Serenity', 'Casablanca'
, 'Amelie'],
                        index=['User 1', 'User 2','User 3', 'User 4', 'User 5', 'User 6', 'User 7'])
print("reconstruction of user-to-movie:")
A_aprox_df
```

```
U of size (7, 3) :
[[-0.1376   0.0236   0.01081]
 [-0.4128   0.07086  0.03244]
 [-0.5503   0.0944   0.04324]
 [-0.688    0.11804  0.05405]
 [-0.1528  -0.5913  -0.654  ]
 [-0.0722  -0.7314   0.678  ]
 [-0.0764  -0.2957  -0.327  ]]
Singular values:
[12.484  9.51   1.346]
as a matrix:
[[12.484  0.     0.   ]
 [ 0.     9.51   0.   ]
 [ 0.     0.     1.346]]
V of size (3, 5) :
[[-0.5625  -0.593   -0.5625  -0.09015 -0.09015]
 [ 0.1266  -0.02878  0.1266  -0.6953  -0.6953 ]
 [ 0.4097  -0.8047   0.4097   0.09125  0.09125]]
reconstruction of user-to-movie:
```

Out[13]:

	Matrix	Alien	Serenity	Casablanca	Amelie
User 1	1.0	1.0	1.0	0.0	0.0
User 2	3.0	3.0	3.0	-0.0	-0.0
User 3	4.0	4.0	4.0	0.0	-0.0
User 4	5.0	5.0	5.0	-0.0	-0.0
User 5	0.0	2.0	-0.0	4.0	4.0
User 6	0.0	0.0	-0.0	5.0	5.0
User 7	0.0	1.0	-0.0	2.0	2.0

- $A = U \Sigma V^T$ - example:

U is "user-to-concept" similarity matrix

$$
\begin{array}{cc}
 & \begin{matrix} \text{Matrix} & \text{Alien} & \text{Serenity} & \text{Casablanca} & \text{Amelie} \end{matrix} \\
\begin{matrix} \text{SciFi} \\ \text{SciFi} \\ \text{SciFi} \\ \text{SciFi} \\ \text{Romnce} \\ \text{Romnce} \\ \text{Romnce} \end{matrix} &
\begin{bmatrix}
1 & 1 & 1 & 0 & 0 \\
3 & 3 & 3 & 0 & 0 \\
4 & 4 & 4 & 0 & 0 \\
5 & 5 & 5 & 0 & 0 \\
0 & 2 & 0 & 4 & 4 \\
0 & 0 & 0 & 5 & 5 \\
0 & 1 & 0 & 2 & 2
\end{bmatrix}
\end{array}
=
\begin{array}{c}
\begin{matrix} \text{SciFi-concept} & \text{Romance-concept} & \end{matrix} \\
\begin{bmatrix}
0.13 & 0.02 & -0.01 \\
0.41 & 0.07 & -0.03 \\
0.55 & 0.09 & -0.04 \\
0.68 & 0.11 & -0.05 \\
0.15 & -0.59 & 0.65 \\
0.07 & -0.73 & -0.67 \\
0.07 & -0.29 & 0.32
\end{bmatrix}
\end{array}
\times
\begin{bmatrix}
12.4 & 0 & 0 \\
0 & 9.5 & 0 \\
0 & 0 & 1.3
\end{bmatrix}
\times
\begin{bmatrix}
0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\
0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\
0.40 & -0.80 & 0.40 & 0.09 & 0.09
\end{bmatrix}
$$

[Great video from Stanford explaining SVD with the same example (https://www.youtube.com/watch?v=P5mlg91as1c)](https://www.youtube.com/watch?v=P5mlg91as1c)

Credits

- Icons from Icon8.com (https://icons8.com/) - https://icons8.com (https://icons8.com)
- Datasets from Kaggle (https://www.kaggle.com/) - https://www.kaggle.com/ (https://www.kaggle.com/)