CS131 Notes

Sean Wu

May 20, 2020

Contents

1 Introduction

1.1 What is Computer Vision and why is it hard

Computer Vision : extracting info from digital images OR developing algorithms to understand image content for other applications

- Computer Vision is a hard interdisciplinary problem that is still unsolved
  - Hard to convert data storing RGB values in many pixels to semantic info (ex. this blob of black pixels is a chair)
- Vision (extracting meaningful info) is harder than 3D modelling

1.2 Definition of Vision and Comparisons to Human Vision

sensing device : captures details from a scene

interpreting device : processes image from sensing device to extract meaning

- Humans use eyes as sensing devices while computers use cameras
- For sensing devices, computer vision is actually better than human vision because cameras can see infrared, have longer range, and capture greater detail
- For interpreting devices, the human brain is way more advanced than computer systems

1.3 Human Vision Strengths and Weaknesses

- Human vision evolved to quickly recognize danger for survival
- It is very fast $\longrightarrow \sim 150$ ms to recognize an animal
- For speed, humans *focus* only on "relevant" *areas of interest*
- Thus, small signals/changes in the background can be difficult to detect and segment
- Humans also use *context* to infer clues
  - Used to determine next area of focus, when to expect certain objects in certain positions, and colour compensation in shadows
  - However, context can be used to trick human vision
- Context is very hard to include in computer vision

1.4 Extracting info from images

- 2 types of info extracted in computer vision: **measurements** and **semantic info**

1.4.1 Measurement in Vision

- Robots scan surroundings to make a map of its environment
- Stereo vision gives depth information (like 2 eyes) using triangulation
  - Depth info represented as a depth map
- With multiple viewpoints of an object, a 3D surface can be created (or even a 3D model)

1.4.2 Obtaining Semantic Info from Vision

- Labelling objects (or scene)
- Recognizing people, actions, gestures, faces

1.5 Applications of Computer Vision

- Video special effects
- 3D object modelling
- Scene recognition
- Face detection
  - Note: face recognition is harder than face detection
- Optical Character Recognition (OCR)
- Reverse image search
- Vision based interaction (ex. Microsoft Kinect)
- Augmented reality
- Virtual reality

2 Linear Algebra Review

2.1 Vectors

- a *column vector* $\mathbf{v} \in \mathbb{R}^{n\times 1}$ where

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \tag{1}$$

- a *row vector* $\mathbf{v}^T \in \mathbb{R}^{1\times n}$ where

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix} \tag{2}$$

- The transpose of a matrix/vector is denoted with a subscript T
- Note: with numpy in python, you can transpose a vector v with v.T

- In 2D and 3D, vectors have a geometric interpretation as points
- Can also use vectors to represent pixels, gradients at an image keypoint, etc
- In this use case, vectors do not have a geometric interpretation, but calculations like "distance" are still useful

– The distance measures "similarity" between 2 vectors

2.2 Matrix

- A *matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$ is an array of numbers with size m by n
- i.e. m rows and n columns

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & a_{d3} & \dots & a_{dn} \end{bmatrix} \tag{3}$$

- if $m = n$, we say that $\mathbf{A}$ is square

2.2.1 Images

- Python represents an *image* as a matrix of pixel brightnesses
- Note: the upper left corner has indices $\underbrace{[x, y]}_{\text{row, column}} = (0, 0)$

– Python indices start at 0

– MATLAB indices start at 1

- Images can be also be represented as a vector of pixels by stacking rows into a single tall column vector

grayscale image : 1 number per pixel; stored as a $m \times n$ matrix

color image : 3 numbers per pixel $\longrightarrow$ red, green, blue brightnesses (RGB); stored as a $m \times n \times 3$ matrix

2.3 Basic Matrix Operations

2.3.1 Addition

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a+1 & b+2 \\ c+3 & d+4 \end{bmatrix} \tag{4}$$

- Can only add matrices with matching dimensions or a scalar

2.3.2 Scaling

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * 3 = \begin{bmatrix} 3a & 3b \\ 3c & 3d \end{bmatrix} \tag{5}$$

2.3.3 Vector Norms

ℓ_1 **Norm - Manhattan Norm** $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_1|$

ℓ_2 **Norm - Euclidean Norm** $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$

ℓ_∞ **Norm - Max Norm** $\|\mathbf{x}\|_\infty = \max_i |x_i|$

ℓ_p **Norm** $\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} x_i^p\right)^{\frac{1}{p}}$

Matrix Norm $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}^2} = \sqrt{\mathrm{tr}(\mathbf{A}^T\mathbf{A})}$

- Note: a matrix norm is a vector norm in a vector space whose elements (vectors) are matrices (of a given dimension)

- Formally, a **norm** is any $f : \mathbb{R}^n \to \mathbb{R}$ that satisfies these 4 properties

1. **Non-negativity**: $\forall \mathbf{x} \in \mathbb{R}^n, f(\mathbf{x}) \geq 0$
2. **Definiteness**: $f(\mathbf{x}) = 0 \iff \mathbf{x} = \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}$

3. **Homogeneity**: $\forall \mathbf{x} \in \mathbb{R}^n, t \in \mathbb{R}, f(t\mathbf{x}) = |t| f(\mathbf{x})$
4. **Triangle Inequality**: $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n, f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$

2.3.4 Inner Product (Dot Product)

- The **inner product (dot product)** $\mathbf{x} \cdot \mathbf{y}$ or $\mathbf{x}^T\mathbf{y}$ is calculated by multiplying the corresponding entries of 2 vectors and adding up the result
- Note: the inner product takes 2 vectors as input and outputs a single scalar

$$\mathbf{x} \cdot \mathbf{y} = |\mathbf{x}||\mathbf{y}| \cos(\theta) \tag{6}$$

$$\mathbf{x}^T\mathbf{y} = \mathbf{x} \cdot \mathbf{y} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i \tag{7}$$

- if $\mathbf{y}$ is a unit vector, then $\mathbf{x} \cdot \mathbf{y} = |\mathbf{x}| \cos(\theta)$ gives the length of $\mathbf{x}$ which lies in the direction of $\mathbf{y}$

2.4 Matrix Multiplication

- Inner dimensions of matrices must match
- For $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, the product $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$ where $\mathbf{C}_{ij} = \sum\limits_{k=1}^{n} A_{ik} B_{kj}$

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} — & a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_m^T & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \dots & b_p \\ | & | & & | \end{bmatrix} = \begin{bmatrix} a_1^T b_1 & a_1^T b_2 & \dots & a_1^T b_p \\ a_2^T b_1 & a_2^T b_2 & \dots & a_2^T b_p \\ \vdots & \vdots & \ddots & \vdots \\ a_m^T b_1 & a_m^T b_2 & \dots & a_m^T b_p \end{bmatrix} \tag{8}$$

- i.e. matrix multiplication gives a matrix where the entries are the dot product of the rows of A and columns B

2.4.1 Properties of Matrix Multiplication

1. **Associative**: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$
2. **Distributive**: $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$
3. **Not Commutative**: Generally, $\mathbf{AB} \neq \mathbf{BA}$

- ex. if $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, then the matrix product $\mathbf{BA}$ does not exist if $m \neq p$

2.5 Matrix Powers

matrix powers : repeated matrix multiplication of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ with itself

$$\mathbf{A}^2 = \mathbf{AA} \qquad \mathbf{A}^3 = \mathbf{AAA} \tag{9}$$

- Note: only *square* matrices can have powers because the dimensions must match

2.6 Matrix Transpose

matrix transpose : flip matrix across the main diagonal so that the rows become the columns, and vice versa

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \tag{10}$$

- Identity: $(\mathbf{ABC})^T = \mathbf{C}^T\mathbf{B}^T\mathbf{A}^T$

2.7 Determinant

determinant : represents the area (or volume) of the parallelogram described by the vectors in the rows of the matrix

- Note: $\det(\mathbf{A})$ takes a matrix input and returns a scalar
- For $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\det(\mathbf{A}) = ad - bc$

2.7.1 Properties of the determinant

1. $\det(AB) = \det(BA)$
2. $\det(A^{-1}) = \frac{1}{\det(1)}$
3. $\det(A^T) = \det(A)$
4. $\det(A) = 0 \iff A$ is singular

2.8 Trace

trace : sum of the main diagonal elements

$$\operatorname{tr}\left(\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}\right) = 1 + 7 = 8 \tag{11}$$

- Note: the $\text{tr}(A)$ is only defined for square matrices
- $\text{tr}(A)$ is invariant to a lot of transformations so it is sometimes used in proofs

2.8.1 Properties of trace

1. $\text{tr}(AB) = \text{tr}(BA)$
2. $\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B)$

2.9 Special Matrices

2.9.1 Identity Matrix

Identity Matrix : a square matrix $\mathbf{I} \in \mathbb{R}^{n \times n}$ with 1's along the main diagonal and 0's everywhere else

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

- For any matrix $\mathbf{A}$ (with proper dimensions)
- $\mathbf{I} \cdot \mathbf{A} = \mathbf{A}$
- $\mathbf{A} \cdot \mathbf{I} = \mathbf{A}$
- i.e. matrix multiplication with $\mathbf{I}$ is commuative (special case)

2.9.2 Diagonal Matrix

Diagonal Matrix : a square matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ with scalars along the diagonal, 0's everywhere else

$$\mathbf{D} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2.5 \end{bmatrix} \tag{13}$$

- For any matrix $\mathbf{B} \in \mathbb{R}^{n \times p}$, $\mathbf{DB}$ scales the rows of $\mathbf{B}$
- Note: the identity matrix $\mathbf{I}$ is a special diagonal matrix that scales all the rows by 1

2.9.3 Symmetric Matrix

Symmetric Matrix : $\mathbf{A}^T = \mathbf{A}$

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 7 \\ 5 & 7 & 1 \end{bmatrix} \tag{14}$$

2.9.4 Skew-symmetric Matrix

Skew-symmetric Matrix : $\mathbf{A}^T = -\mathbf{A}$

$$\begin{bmatrix} 0 & -2 & -5 \\ 2 & 0 & -7 \\ 5 & 7 & 0 \end{bmatrix} \tag{15}$$

2.10 Transformation Matrices

Matrix transformation : transforms vectors by matrix multiplication: $\mathbf{Ax} = \mathbf{x}'$

2.10.1 Scaling Transformation

Scaling matrix : scales components of vector

$$\underbrace{\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}}_{\text{Scaling Matrix}} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} S_x x \\ S_y y \end{bmatrix} \tag{16}$$

2.10.2 Converting to a rotated reference frame

Rotation Matrix : matrix that describes a rotation of a vector or equivalently changing to a rotated reference frame

- i.e. have the same data point but represent it in a new rotated frame
- Note: rotating a reference frame left == rotating a data point to the right
- Recall: a 2D vector stores a component in the x-direction and a component in the y-direction
- Thus the transformation for $\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix}$ is found by computing the dot product of the original vector with the new unit vectors for the x'-direction and y'-direction
- Thus, the new coordinates $\begin{bmatrix} x' \\ y' \end{bmatrix}$ represent the length of the original vector lying in the direction of the new x-, y- axes

- Equivalently, can express the original x-, y- unit vectors in terms of the new x'-, y- unit vectors

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \text{(new x'-axis)} \\ \text{(new y'-axis)} \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} \tag{17}$$

$$= \begin{bmatrix} (\hat{x} \text{ in new x'-,y'- axes}) & (\hat{y} \text{ in new x'-,y'- axes}) \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\mathbf{R}}_{2 \times 2 \text{ Rotation Matrix}} * \begin{bmatrix} x \\ y \end{bmatrix} \tag{19}$$

$$\mathbf{P}' = \mathbf{RP} \tag{20}$$

2.10.3 2D Rotation Matrix

- For a CCW rotation of a point (aka a CW rotation of ref. frame)

$$x' = x\cos(\theta) - y\sin(\theta) \tag{21}$$

$$y' = x\sin(\theta) + y\cos(\theta) \tag{22}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{23}$$

$$\mathbf{P}' = \mathbf{RP} \tag{24}$$

- Note: transpose of a rotation matrix produces a rotation in the opposite direction

2.10.4 Normal Matrices

- Note: $\mathbf{R}$ belongs to the category of **normal** matrices
- Properties of normal matrices

1. $\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$
2. $\det \mathbf{R} = 1$

- Rows of a rotation matrix are always mutually perpendicular (aka orthogonal) unit vectors
- Same with columns

2.10.5 Multiple Transformation Matrices

- For multiple transformation matrices, the transformations are applied one by one from **right to left**

$$\mathbf{P}' = \mathbf{R}_2\mathbf{R}_1\mathbf{SP} \tag{25}$$

$$\mathbf{P}' = (\mathbf{R}_2(\mathbf{R}_1(\mathbf{SP}))) \tag{26}$$

- By associativity, the result is the same as multiplying the matrices first to form a single transformation matrix

$$\mathbf{P}' = (\mathbf{R}_2\mathbf{R}_1\mathbf{S})\mathbf{P} \tag{27}$$

- In general, matrix multiplication allows us to linearly combine components of a vector
- This is sufficient for scaling, rotating, skewing, but we <u>cannot</u> add a constant (not a linear operation)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cd + dy \end{bmatrix} \tag{28}$$

2.11 Homogenous System

2.11.1 Translation

- Hacky Fix: can add translation by representing the problem in a higher $n + 1$ dimension and stick a 1 at the end of every vector

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix} \tag{29}$$

- Note: $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ and $\begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$ are **homogenous coordinates**
- Now we can rotate, scale, skew, and translate
- Matrix multiplication with translation matrix results in adding the rightmost column of the translation vector to the original vector
- Generally, homogenous transformation matries have a bottom row of $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$ so that the resulting vector $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$ has a 1 at the bottom too

2.11.2 Division

- ex. want to divide a vector by a coordinate y_0 to make things scale down as they get farther away in a camera image
- Matrix multiplication can't actually divide so use this convention
- **Convention**: in homogenous coordinates, divide the resulting vector by its last coordinates after matrix multiplication

$$\begin{bmatrix} x \\ y \\ 7 \end{bmatrix} \longrightarrow \begin{bmatrix} \frac{x}{7} \\ \frac{y}{7} \\ 1 \end{bmatrix} \tag{30}$$

2.11.3 2D translation using Homogenous Coordinates

- $P = (x, y) \to (x, y, 1)$
- $T = (t_x, t_y) \to (t_x, t_y, 1)$

$$\mathbf{P}' = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ 0 & 1 \end{bmatrix} * \mathbf{P} \tag{31}$$

- Thus $\mathbf{P}' = \mathbf{T} \cdot \mathbf{P}$ where $\mathbf{T}$ is the translation matrix

2.11.4 Scaling Matrix in Homogenous Coordinates

- $P = (x, y) \to (s_x x, s_y y, 1)$
- $T = (t_x, t_y) \to (t_x, t_y, 1)$
- $P' = (x + t_x, y + t_y) \to (x + t_x, y + t_y, 1)$

$$\mathbf{P}' = \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{S}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}' & 0 \\ 0 & 1 \end{bmatrix} * \mathbf{P} \tag{32}$$

- Thus $\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$ where $\mathbf{S}$ is the scaling matrix

2.11.5 Scaling and Translating

- Recall: matrix transformations are applied right to left for $\mathbf{P}'' = \mathbf{TSP}$

$$\mathbf{P}'' = \mathbf{TSP} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}' & \mathbf{t}' \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix} \tag{33}$$

2.11.6 Scaling & Translating != Translating & Scaling

- Recall: matrix multiplication is generally **not** commutative, so order matters
- If you scale after you translated, both the original vector and the translation will be scaled

2.11.7 Rotation Matrix in Homogenous Coordinates

- Rotation $\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$ in homogenous coordinates is the same as regular rotation, just with the extra 1 in the bottom row

$$\mathbf{P}' = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}' & 0 \\ 0 & 1 \end{bmatrix} * \mathbf{P} \tag{34}$$

2.11.8 Scaling + Rotation + Translation

$$\mathbf{P}' = (\mathbf{TRS})\mathbf{P} \tag{35}$$

$$= \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{S} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{36}$$

$$= \begin{bmatrix} \mathbf{RS} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{37}$$

- Therefore, the **general transformation matrix** is $\begin{bmatrix} \mathbf{RS} & \mathbf{t} \\ 0 & 1 \end{bmatrix}$

2.12 Matrix Inverse

- Given an invertible matrix $\mathbf{A}$, its inverse $\mathbf{A}^{-1}$ is a matrix such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$
- ex. $\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$
- The inverse $\mathbf{A}^{-1}$ doesn't always exist
- If $\mathbf{A}^{-1}$ exists, $\mathbf{A}$ is **invertible** (aka **nonsingular**)
- Otherwise, it is **non-invertible/singular**

2.12.1 Properties of the Matrix Inverse

1. $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$
2. $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$
3. $\mathbf{A}^{-T} \triangleq (\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$

2.12.2 Pseudo Inverse

- if inverse $\mathbf{A}^{-1}$ exists, we can solve $\mathbf{Ax} = \mathbf{b}$ with $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$

```
np.linalg.inv(A) * b
```

- If inverse $\mathbf{A}^{-1}$ doesn't exist or the matrix is too large (too expensive to compute), we can use the pseudo-inverse to find $\mathbf{x}$

```
np.linalg.solve(A,b)
```

- Python will try several numerical methods (including pseudoinverse) and return solution for $\mathbf{x}$
- if no exact solution $\longrightarrow$ Python returns the closest value
- if many solutions $\longrightarrow$ Python returns the smallest one

2.13 Linear Independence

- For a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$, if we can express $\mathbf{v}_1$ as a **linear combination** of other vectors $\mathbf{v}_2, \ldots, \mathbf{v}_n$, then $\mathbf{v}_1$ is **linearly dependent** on the other vectors
- ex. $\mathbf{v}_1 = 0.7\mathbf{v}_2 - 0.7\mathbf{v}_n$

Linearly independent set : no vector in a set is linearly dependent on the rest of the vectors

- ex. a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is always linearly independent if each vector is perpendicular to every other vector (and nonzero)

2.14 Matrix Rank

Rank : the rank of a transformation matrix tells you how many dimensions it transforms a vector to; i.e. the dimensions of the output vecor

col-rank : number of linearly independent column vectors of $\mathbf{A}$

row-rank : number of linearly independent row vectors of $\mathbf{A}$

- Note: column rank always equals row rank

$$\text{rank}(\mathbf{A}) \triangleq \text{col-rank}(\mathbf{A}) = \text{row-rank}(\mathbf{A}) \tag{38}$$

- ex. if $\text{rank}(\mathbf{A}) = 1$, then the transformation $\mathbf{P}' = \mathbf{AP}$ maps points onto a line

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + y \\ 2x + 2y \end{bmatrix} \tag{39}$$

- Here all the points are mapped to the line $y = 2x$

full rank : if an $m \times m$ matrix has rank m, we say it is full rank. It maps an $m \times 1$ vector uniquely to another $m \times 1$ vector. Also has an inverse matrix

singular : if an $m \times m$ matrix has rank $< m$, then at least one dimension is getting collapsed to zero. Thus there is no way to look at the output and find the input (not invertible)

- If an $m \times m$ matrix has **full rank** $\iff$ it is invertible

2.15 Eigenvector & Eigenvalues

Eigenvector : an eigenvector $\mathbf{x}$ of a linear transformation $\mathbf{A}$ is a nonzero vector that when $\mathbf{A}$ is applied to it, does not change direction

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \qquad \mathbf{x} \neq 0 \tag{40}$$

- Applying $\mathbf{A}$ to an eigenvector only scales the eigenvector by the scalar value λ, called an **eigenvalue**
- An $m \times m$ matrix will have $\leq m$ eigenvectors where the eigenvalue λ is nonzero
- To find all eigenvalues of $\mathbf{A}$ solve this eqn for $\mathbf{x} \neq 0$

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{41}$$

$$\mathbf{A}\mathbf{x} = (\lambda\mathbf{I})\mathbf{x} \tag{42}$$

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0 \tag{43}$$

- Since $\mathbf{x} \neq 0$, $(\mathbf{A} - \lambda\mathbf{I})$ cannot be invertible/nonsingular and its determinant is zero (i.e. nonzero nullspace)

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0 \tag{44}$$

2.15.1 Properties of Eigenvectors and Eigenvalues

1. The trace of $\mathbf{A}$ is the sum of its eigenvalues

$$\mathrm{tr}(\mathbf{A}) = \sum_{i=1}^{n} \lambda_i \tag{45}$$

2. The determinant of $\mathbf{A}$ equal to product of its eigenvalues

$$\det(\mathbf{A}) = \prod_{i=1}^{n} \lambda_i \tag{46}$$

3. The rank of $\mathbf{A}$ is equal to the number of non-zero eigenvalues
4. Eigenvalues of a diagonal matrix $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ are just the diagonal entries $d_1, \ldots, d_n$

2.15.2 Spectral Theory

eigenpair : an eigenvalue λ and its associated eigenvector $\mathbf{x}$

eigenspace : the eigenspace associated with eigenvalue λ is the space of vectors where $\mathbf{A} - \lambda\mathbf{I} = 0$

spectrum of A : the set of all eigenvalues of a matrix $\mathbf{A}$

$$\sigma(\mathbf{A}) = \{\lambda \in \mathbb{C} \mid \det(\mathbf{A} - \lambda\mathbf{I}) = 0\} \tag{47}$$

spectral radius : magnitude of the largest eigenvalue

$$\rho(\mathbf{A}) = \max\{|\lambda_1|, \ldots, |\lambda_n|\} \tag{48}$$

Theorem 1 (Spectral radius bound)**.** *Spectral radius is bounded by the infinity norm of a matrix*

$$\rho(\mathbf{A}) = \lim_{k\to\infty} \left\|\mathbf{A}^k\right\|^{\frac{1}{k}} \tag{49}$$

Proof. let $|\lambda|^k \|\mathbf{v}\| = \left\||\lambda|^k \mathbf{v}\right\| = \left\|\mathbf{A}^k \mathbf{v}\right\|$

By the triangle rule,

$$|\lambda|^k \|\mathbf{v}\| \leq \left\|\mathbf{A}^k\right\| \cdot \|\mathbf{v}\| \tag{50}$$

and since $\mathbf{v} \neq 0$

$$|\lambda|^k \leq \left\|\mathbf{A}^k\right\| \tag{51}$$

which gives us

$$\rho(\mathbf{A}) = \lim_{k\to\infty} \left\|\mathbf{A}^k\right\|^{\frac{1}{k}} \tag{52}$$

■

2.16 Diagonalization

- A $n \times n$ matrix $\mathbf{A}$ is diagonalizable if it has n linearly independent eigenvectors
- Most square matrices are diagonalizable
  - Normal matrices are diagonalizable
  - Matrices w/ n distinct eigenvectors are diagonalizable

Lemma 1. *Eigenvectors associated with distinct eigenvalues are linearly independent*

- Eigenvalue equation can be written as $\mathbf{AV} = \mathbf{VD}$

- $\mathbf{D}$ is the matrix of eigenvalues and $\mathbf{V}$ is the matrix of corresponding eigenvectors

$$\mathbf{D} = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \tag{53}$$

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \ldots \mathbf{v}_n \end{bmatrix} \tag{54}$$

- Assuming all λ_i's are unique, can diagonalize $\mathbf{A}$ by $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}$
- Recall: eigenvectors are independent so $\mathbf{V}$ is invertible
- if the eigenvectors are also all mutually orthogonal, then $\mathbf{V}$ is an orthogonal matrix and its inverse is its transpose so $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T$

2.16.1 Symmetric Matrices

- if $\mathbf{A}$ is symmetric, then all of its eigenvalues are real and its eigenvectors are orthonormal
- So we can diagonalize $\mathbf{A}$ by $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T$
- Given $y = \mathbf{V}^T x$

$$x^T \mathbf{A} x = x^T \mathbf{V}\mathbf{D}\mathbf{V}^T x = y^T \mathbf{D} y = \sum_{i=1}^{n} \lambda_i y_i^2 \tag{55}$$

- Thus, for the following maximization

$$\max_{x \in \mathbb{R}^n} x^T \mathbf{A} x \text{ subject to } \|x\|_2^2 = 1 \tag{56}$$

- Then the maximizing x can be found by finding the eigenvector that corresponds to the largest eigenvalue of $\mathbf{A}$

2.16.2 Applications of Eigenvalues and Eigenvectors

1. PageRank
2. Schrodinger equation
3. Principle Component Analysis (PCA)
4. Image compression

2.17 Matrix Calculus

2.17.1 Gradient

- Let a function $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ take as input a matrix $A \in \mathbb{R}^{m \times n}$ and returns a real value

- Then the **gradient of f** is

$$\nabla_{\mathbf{A}} f(\mathbf{A}) = \begin{bmatrix} \frac{\partial f(\mathbf{A})}{\partial A_{11}} & \frac{\partial f(\mathbf{A})}{\partial A_{12}} & \cdots & \frac{\partial f(\mathbf{A})}{\partial A_{1n}} \\ \frac{\partial f(\mathbf{A})}{\partial A_{21}} & \frac{\partial f(\mathbf{A})}{\partial A_{22}} & \cdots & \frac{\partial f(\mathbf{A})}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(\mathbf{A})}{\partial A_{m1}} & \frac{\partial f(\mathbf{A})}{\partial A_{m2}} & \cdots & \frac{\partial f(\mathbf{A})}{\partial A_{mn}} \end{bmatrix} \tag{57}$$

- Every entry in the matrix is

$$\nabla_{\mathbf{A}} f(\mathbf{A})_{ij} = \frac{\partial f(\mathbf{A})}{\partial A_{ij}} \tag{58}$$

- The size of $\nabla_{\mathbf{A}} f(\mathbf{A})$ is always the same size as $\mathbf{A}$
- So if $\mathbf{A}$ is just a vector $\mathbf{x}$, then

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{59}$$

- ex. for $\mathbf{x} \in \mathbb{R}^n$, let $f(\mathbf{x}) = \mathbf{b}^T\mathbf{x}$ for some known vector $\mathbf{b} \in \mathbb{R}^n$

$$f(\mathbf{x}) = \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} b_i x_i \tag{60}$$

$$\frac{\partial f(\mathbf{x})}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_{i=1}^{n} b_i x_i = b_k \tag{61}$$

$$\therefore \nabla_{\mathbf{x}} \mathbf{b}^T \mathbf{x} = \mathbf{b} \tag{62}$$

2.17.2 Properties of the Gradient

1. $\nabla_{\mathbf{x}}(f(\mathbf{x}) + g(\mathbf{x})) = \nabla_{\mathbf{x}} f(\mathbf{x}) + \nabla_{\mathbf{x}} g(\mathbf{x})$
2. For $t \in \mathbb{R}$, $\nabla_{\mathbf{x}}(tf(\mathbf{x})) = t\nabla_{\mathbf{x}} f(\mathbf{x})$

2.18 Hessian Matrix

- The **Hessian matrix** with respect to the vector $\mathbf{x} \in \mathbb{R}^n$ can be written as $\nabla^2_{\mathbf{x}} f(\mathbf{x})$ or as $\mathbf{H}$ and is an $n \times n$ matrix of partial derivatives

$$\nabla^2_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix} \tag{63}$$

- Each entry is

$$\nabla^2_{\mathbf{x}} f(\mathbf{x})_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \tag{64}$$

- Note: Hessian is the gradient **of every** entry of the gradient of the vector
- ex. 1^{st} column of the Hessian is the gradient of $\frac{\partial f(\mathbf{x})}{\partial x_1}$
- Note: Hessian is always symmetric because of Schwarz's Theorem

Theorem 2 (Schwarz's Theorem)**.**

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} = \frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_i} \tag{65}$$

Order of partial derivatives doesn't matter as long as the 2^{nd} derivative exists and is continuous

- ex. Consider quadratic function $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}$

$$f(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j \tag{66}$$

$$\begin{aligned} \frac{\partial f(\mathbf{x})}{\partial x_k} &= \frac{\partial}{\partial x_k} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j & (67) \\ &= \frac{\partial}{\partial x_k} \left[\sum_{i \neq k} \sum_{j \neq k} A_{ij} x_i x_j + \sum_{i \neq k} A_{ik} x_i x_k + \sum_{j \neq k} A_{kj} x_k x_j + A_{kk} x_k^2 \right] & (68) \\ &= \sum_{i \neq k} A_{ik} x_i + \sum_{j \neq k} A_{kj} x_j + 2 A_{kk} x_k & (69) \\ &= \sum_{i=1}^{n} A_{ik} x_i + \sum_{j=1}^{n} A_{kj} x_j = 2 \sum_{i=1}^{n} A_{ki} x_i & (70) \end{aligned}$$

$$\begin{aligned} \frac{\partial^2 f(\mathbf{x})}{\partial x_k \partial x_l} &= \frac{\partial}{\partial x_k} \left[\frac{\partial f(\mathbf{x})}{\partial x_l} \right] = \frac{\partial}{\partial x_k} \left[\sum_{i=1}^{n} 2 A_{li} x_i \right] & (71) \\ &= 2 A_{lk} = 2 A_{kl} & (72) \end{aligned}$$

- Thus $\nabla^2_{\mathbf{x}} f(\mathbf{x}) = 2\mathbf{A}$

2.19 Singular Value Decomposition

- Several computer algorithms can "factorize" a matrix into the product of other matrices
- Singular Value Decomposition is the most useful

Singular Value Decomposition (SVD) : represent a matrix $\mathbf{A}$ as a product of 3 matrices $\mathbf{U}$, $\mathbf{S}$, $\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}^T$ are rotation matrices and $\mathbf{S}$ is a scaling matrix

- MATLAB: [U,S,V] = **svd**(A)
- ex.

$$\underbrace{\begin{bmatrix} -0.40 & 0.916 \\ -0.916 & 0.40 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix}}_{\mathbf{S}} \underbrace{\begin{bmatrix} -0.05 & 0.999 \\ 0.999 & 0.05 \end{bmatrix}}_{\mathbf{V}^T} = \underbrace{\begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}}_{\mathbf{A}} \tag{73}$$

- In general, if $\mathbf{A}$ is $m \times n$, then $\mathbf{U}$ will be $m \times m$, $\mathbf{S}$ will be $m \times n$ and $\mathbf{V}^T$ will be $n \times n$
- ex.

$$\underbrace{\begin{bmatrix} -0.39 & -0.92 \\ -0.92 & 0.39 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & 0.77 & 0 \end{bmatrix}}_{\mathbf{S}} \underbrace{\begin{bmatrix} -0.42 & -0.57 & -0.70 \\ 0.81 & 0.11 & -0.58 \\ 0.41 & -0.82 & 0.41 \end{bmatrix}}_{\mathbf{V}^T} = \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}_{\mathbf{A}} \tag{74}$$

- Note: $\mathbf{U}$ and $\mathbf{V}$ are always rotation matrices
  - also called "unitary" matrices because each column is a unit vector
- $\mathbf{S}$ is a diagonal matrix whose number of nonzero entries is the rank $\mathbf{A}$

2.19.1 SVD Applications

- Each product of (column i of $\mathbf{U}$) $\cdot$ (value i from $\mathbf{S}$) $\cdot$ (row i of $\mathbf{V}^T$) produces a component of the final $\mathbf{A}$
- We are building $\mathbf{A}$ as a linear combination of the columns of $\mathbf{U}$
- If we use all columns of $\mathbf{U}$, we can rebuild the original $\mathbf{A}$ perfectly
- But with real-world data, we can often just use the first few columns of $\mathbf{U}$ and get something close to $\mathbf{A}$
- Thus we call the first few columns of $\mathbf{U}$ the **Principal Components** of the data
- Principal components show the major patterns that can be added together to produce the columns of the original matrix
- Rows of $\mathbf{V}^T$ show how the principal components are mixed to produce the columns of $\mathbf{A}$
- For SVD with images, can use first few principal components to reproduce a recognizable picture

2.19.2 Principal Component Analysis

- Recall: columns of $\mathbf{U}$ are the Principal Components of the data

Principal Component Analysis (PCA) : construct a matrix $\mathbf{A}$ where each column is a separate data sample. Run SVD on $\mathbf{A}$ and look at the first few columns of $\mathbf{U}$ to see the common patterns

- Often raw data can have a lot of redundancy and patterns
- PCA allows you to represent data samples as weights on the principal components, rather than using the original raw form of the data
- This minimal PCA representation makes machine learning and other algorithms much more efficient

2.19.3 SVD Algorithm

- Computers can find eigenvectors $\mathbf{x}$ such that $\mathbf{Ax} = \lambda\mathbf{x}$ using this iterative algorithm

```
x = random unit vector
while (x not converged)
  x = Ax
  normalize x
```

- $\mathbf{x}$ will quickly converge to an eigenvector
- Some adjustments let this algorithm find all eigenvectors
- Note: eigenvectors are for square matrices, but SVD is for all matrices
- To do **svd**(A), computers do this

1. Take eigenvectors of $\mathbf{AA}^T$
   - These eigenvectors are the columns of $\mathbf{U}$
   - Square root of eigenvalues are the singular values (the entries of $\mathbf{S}$)
2. Take eigenvectors of $\mathbf{A}^T\mathbf{A}$
   - These eigenvectors are columns of $\mathbf{V}$ (or rows of $\mathbf{V}^T$)

- SVD is fast (even for large matrices)