BASIC TECHNIQUES FOR COMPUTER SIMULATIONS WPO – 3RD SESSION

MATRICES

VRIJE UNIVERSITEIT BRUSSEL

INTRODUCTION

- A homogeneous linear algebraic system:

$$\left.\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = 0 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = 0 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = 0 \end{array}\right\}$$

$$\mathbf{A}\mathbf{x} = 0$$

Trivial solution: $\mathbf{x} = 0$

a: constant coefficients, x: unknowns

- To find nontrivial solutions ($\mathbf{x} \neq 0$):

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

$\boldsymbol{\lambda}$ is an eigenvalue, $\mathbf{x}$ is its associated eigenvector, and $\mathbf{I}$ is the identity matrix

INTRODUCTION – PHYSICAL EXPLANATION OF EIGENVALUES

- $\mathbf{Ax} = 0$ when considering a system in a passive, **equilibrium** state (e.g. bridge standing still with no external forces)
- $(\mathbf{A}-\lambda\mathbf{I})\mathbf{x} = 0$ when exploring a system's **intrinsic** characteristics, responses to **perturbations**, **dynamic** properties

Eigenvalues can represent:

- Natural frequencies of a system (mechanical vibrations);
- Stability of a system (control theory);
- Energy levels (quantum mechanics);
- Transition to instabilities (fluid dynamics);
- Reaction rates (chemical kinetics);
- Growth/decay rates (population dynamics);
- …

POLYNOMIAL METHOD

- The eigenvalues are the roots of the characteristic polynomial:

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

e.g.: (2x2)

$$\begin{vmatrix} (a_{11}-\lambda) & a_{12} \\ a_{21} & (a_{22}-\lambda) \end{vmatrix} = \underbrace{\lambda^2 - (a_{11}+a_{22})\lambda - a_{12}a_{21}}_{\text{characteristic polynomial}} = 0 \rightarrow \begin{cases} \lambda_1 = \dots \\ \lambda_2 = \dots \end{cases}$$

By substituting λ1 and λ2 into the system we find the corresponding eigenvectors

VRIJE UNIVERSITEIT BRUSSEL

POLYNOMIAL METHOD: AN EXAMPLE

- 2x2 homogeneous system: $10x_1 - 5x_2 = 0$
  $-5x_1 + 10x_2 = 0$

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

$$\begin{vmatrix} (10-\lambda) & -5 \\ -5 & (10-\lambda) \end{vmatrix} = (10-\lambda)^2 - 25 = 0 \rightarrow \begin{cases} \lambda_1 = 15 \\ \lambda_2 = 5 \end{cases}$$

(eigenvalues)

POLYNOMIAL METHOD: AN EXAMPLE

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

- For $\lambda=\lambda_1=15$: $\left.\begin{array}{l} -5x_1 - 5x_2 = 0 \\ -5x_1 - 5x_2 = 0 \end{array}\right\}$ $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ eigenvector

- For $\lambda=\lambda_2=5$: $\left.\begin{array}{l} 5x_1 - 5x_2 = 0 \\ 5x_1 - 5x_2 = 0 \end{array}\right\}$ $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ eigenvector

Note: A correct **eigenvalue will make both equations identical**. This means that there are infinite solutions. **Eigenvectors show that the ratio** of the unkowns is **constant**.

FINDING EIGENVALUES AND EIGENVECTORS IN PYTHON

- From the previous example: $\mathbf{A} = \begin{bmatrix} 10 & -5 \\ -5 & 10 \end{bmatrix}$

l, v = numpy.linalg.eig(A)

Vector **l** contains the eigenvalues,
vector **v** contains the eigenvectors (one eigenvector <u>per column</u>)

<u>Note</u>: The eigenvectors may seem different depending on the method we use to derive them, but this is because they are scaled differently. In any case, their ratios must be identical.

THE POWER METHOD

- Iterative method to determine the largest (or dominant) eigenvalue, and the corresponding eigenvector of a square matrix **A**.

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

$$\begin{aligned} 10x_1 - 5x_2 &= 0 \\ -5x_1 + 10x_2 &= 0 \end{aligned}$$

Step 1: Initial guess for $\mathbf{x_0}$, with largest entry equal to 1

$$\boldsymbol{x_0} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Step 2: • $\mathbf{p_0} = \mathbf{Ax_0}$

- n_0 is the maximum absolute value of $\mathbf{p_0}$
- scale and determine the next guess for **x**: $x_1 = \frac{p_0}{n_0}$
- loop until convergence criterion is met: $\left|\frac{n_{k+1} - n_k}{n_{k+1}}\right| < tol$

Step 3: After convergence, maximum eigenvalue: $\lambda_{max}=n_{k+1}$ and corresponding eigenvector: $\mathbf{v}=\mathbf{x_{k+1}}$

THE INVERSE POWER METHOD

- Iterative method to determine the smallest eigenvalue, and the corresponding eigenvector.

The only difference to power method: work with $\mathbf{A}^{-1}$, instead of $\mathbf{A}$

The method will output the maximum eigenvalue of $\mathbf{A}^{-1}$, which is: $\frac{1}{\lambda_{min}}$

VRIJE UNIVERSITEIT BRUSSEL

SINGULAR VALUE DECOMPOSITION (SVD)

- Decomposition of a matrix **A** into three matrices:

$$\mathbf{A} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^{\mathsf{T}}$$

U: left singular vectors, **Σ**: singular values, **V**: right singular vectors

$$\begin{bmatrix} | & | & & | \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & \cdots & \boldsymbol{u}_m \\ | & | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & & 0 \\ 0 & \sigma_2 & & 0 \\ 0 & 0 & \cdots & \sigma_n \\ 0 & 0 & & 0 \\ 0 & 0 & & 0 \\ 0 & 0 & & 0 \end{bmatrix} \begin{bmatrix} | & | & & | \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n \\ | & | & & | \end{bmatrix}^{\mathsf{T}}$$

(mxn) (mxm) (mxn) (nxn)

SVD PROPERTIES

$$\mathbf{A} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^{T}$$

U and **V**: square, unitary and orthogonal. So: $\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$

$\mathbf{V}\mathbf{V}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$

Σ: diagonal and non-negative, hierarchically ordered. So: $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_n \geq 0$

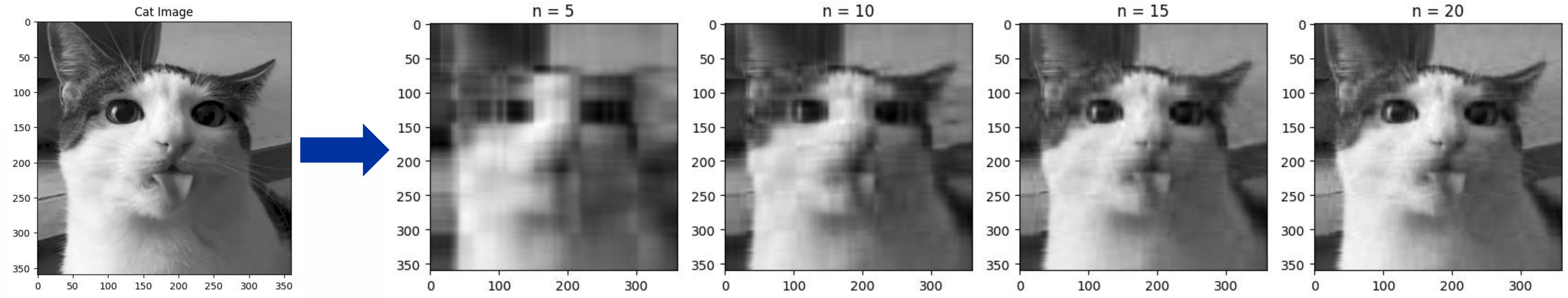
SVD APPLICATIONS

The decomposition can reveal important geometric and algebraic properties about **A**. It is used for:

- Dimensionality Reduction (Data analysis)
- Noise reduction (Signal processing)
- Image compression (Image processing)
- Solution of linear equations
- …

SVD: IMAGE COMPRESSION

We can upload on python an image and represent it as a matrix of integers, with each integer representing the brightness of the pixel in its position



Source: https://dmicz.github.io/machine-learning/svd-image-compression/

$$\mathbf{A} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^{\mathbf{T}} = \mathbf{u_1}\sigma_1\mathbf{v_1}^{T} + \mathbf{u_2}\sigma_2\mathbf{v_2}^{T} + \ldots + \mathbf{u_n}\sigma_n\mathbf{v_n}^{T}$$

We can choose how many terms (or singular values) to keep, in order to reconstruct **A**

SVD: HOW TO COMPUTE THE MATRICES

$\mathbf{A} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^T$

- $\mathbf{AA}^T$ and $\mathbf{A}^T\mathbf{A}$ are symmetric matrices: they have real, positive, and equal eigenvalues

PYTHON

- The eigenvectors of $\mathbf{AA}^T$ form the columns of **U** — numpy.linalg.eig(AA^T)
- The eigenvectors of $\mathbf{A}^T\mathbf{A}$ form the columns of **V** — numpy.linalg.eig(A^TA)
- Singular values of $\boldsymbol{\Sigma}$: $\sigma_i = \sqrt{\lambda_i}$ in hierarchical order $(\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_n \geq 0)$ where λ_i: eigenvalues of $\mathbf{AA}^T$ (and $\mathbf{A}^T\mathbf{A}$)

Cross-check your results with: numpy.linalg.svd(A)

QR DECOMPOSITION (I)

- Decomposition of a matrix **A** into two matrices:

$$\mathbf{A} = \mathbf{Q}\,\mathbf{R}$$

Q: orthogonal matrix, **R**: upper triangular matrix

Q and **R** can be determined numerically with the Gram-Schmidt process:

1st Gram-Schmidt vector : $\mathbf{A_1} = \mathbf{a_1}$ and $q_1 = \dfrac{A_1}{\|A_1\|} = \dfrac{a_1}{\|a_1\|}$

2nd Gram-Schmidt vector : project $\mathbf{a_2}$ on $\mathbf{q_1}$ and substract it from $\mathbf{a_2}$:

$$\mathbf{A_2} = \mathbf{a_2} - (\mathbf{a_2}\mathbf{q_1})\mathbf{q_1}$$

$$q_2 = \frac{A_2}{\|A_2\|}$$

$$A = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & & | \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_n \\ | & | & & | \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} a_1q_1 & a_2q_1 & \cdots & a_nq_1 \\ 0 & a_2q_2 & \cdots & a_nq_2 \\ & \vdots & & \\ 0 & 0 & \dots & a_nq_n \end{bmatrix}$$

QR DECOMPOSITION (II)

3rd Gram-Schmidt vector : project $\mathbf{a_3}$ on $\mathbf{q_1}$ and $\mathbf{q_2}$ and substract it from $\mathbf{a_3}$:

$$\mathbf{A_3} = \mathbf{a_3} - (\mathbf{a_3}\,\mathbf{q_1})\mathbf{q_1} - (\mathbf{a_3}\,\mathbf{q_2})\mathbf{q_2} \quad \text{and} \quad q_3 = \frac{A_3}{\|A_3\|}$$

…

(k+1)th Gram-Schmidt vector: $\mathbf{A_{k+1}} = \mathbf{a_{k+1}} - (\mathbf{a_{k+1}}\mathbf{q_1})\mathbf{q_1} - (\mathbf{a_{k+1}}\mathbf{q_2})\mathbf{q_2} - \ldots - (\mathbf{a_{k+1}}\,\mathbf{q_k})\mathbf{q_k}$ and $q_{k+1} = \frac{A_{k+1}}{\|A_{k+1}\|}$

$$\mathbf{R} = \mathbf{Q^T}\,\mathbf{A} = \begin{bmatrix} - & q_1 & - \\ - & q_2 & - \\ & | & \\ - & q_n & - \end{bmatrix} \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & & | \end{bmatrix}$$

QR ALGORITHM TO FIND EIGENVALUES

- The following iterations of the QR decomposition can be used to compute eigenvalues of a <u>square</u> matrix **A**:

1) $\mathbf{A} = \mathbf{QR}$ (QR decomposition)

2) Form $\mathbf{A_1} = \mathbf{RQ}$ (A_1 is a similar to A: same eigenvalues)
$\mathbf{A_1} = \mathbf{Q_1R_1}$ (QR decomposition)

3) Form $\mathbf{A_2} = \mathbf{R_1Q_1}$
$\mathbf{A_2} = \mathbf{Q_2R_2}$ (QR decomposition)

. . .

Repeat until convergence:
$|\mathbf{A_{k+1}} - \mathbf{A_k}| < \mathrm{tol}$
for all columns

The diagonal values of the final $\mathbf{A_{k+1}}$ contain the eigenvalues of **A**