UNIVERSITY OF MINNESOTA TWIN CITIES

# *Numerical Linear Algebra for Machine Learning*

*Yousef Saad*
University of Minnesota

UM6P Spring School 2025
Benguerir, Morocco

Feb. 17-25, 2025

# *This tutorial: Topics & Plan*

➤ Tutorial on: Numerical Linear Algebra for Machine Learning with emphasis on Graph-based methods

- *First* part: background in linear algebra, sparse matrices, graphs
- *Second:* data-related problems: unsupervised learning, dimension reduction, embeddings, ..
- *Third:*: Deep learning, graph neural networks,
- Hands-on practice and demos [in matlab and Python+pytorch]
- Prerequisite: senior level course in numerical linear algebra
- All materials posted here:

*https://www.cs.umn.edu/~saad/talks.html*

# *(Rough) Schedule*

| | |
|---|---|
| *Feb. 17* | General introduction; Background & Examples; Eigenvalue Pbs; Projection methods; The SVD; Sparse matrices; Data structures; Review: Graphs; Graphs & sparse matrices. |
| *Feb. 21* | Graph centrality; Graph Laplaceans; Clustering; Dimension reduction; From Data to Graphs; Networks & Centrality; Graph Laplaceans; Clustering; Segmentation. |
| *Feb. 24* | Graph embedding; Deep Neural Networks; Attention; Transformers; Graph-based methods; Graph Neural Networks; GCN; GAT; Graph Coarsening [if time permits] |

# GENERAL INTRODUCTION AND BACKGROUND

# *Example of a classical problem ('The old'): Fluid flow*

*Physical Model*

↓

*Nonlinear PDEs*

↓

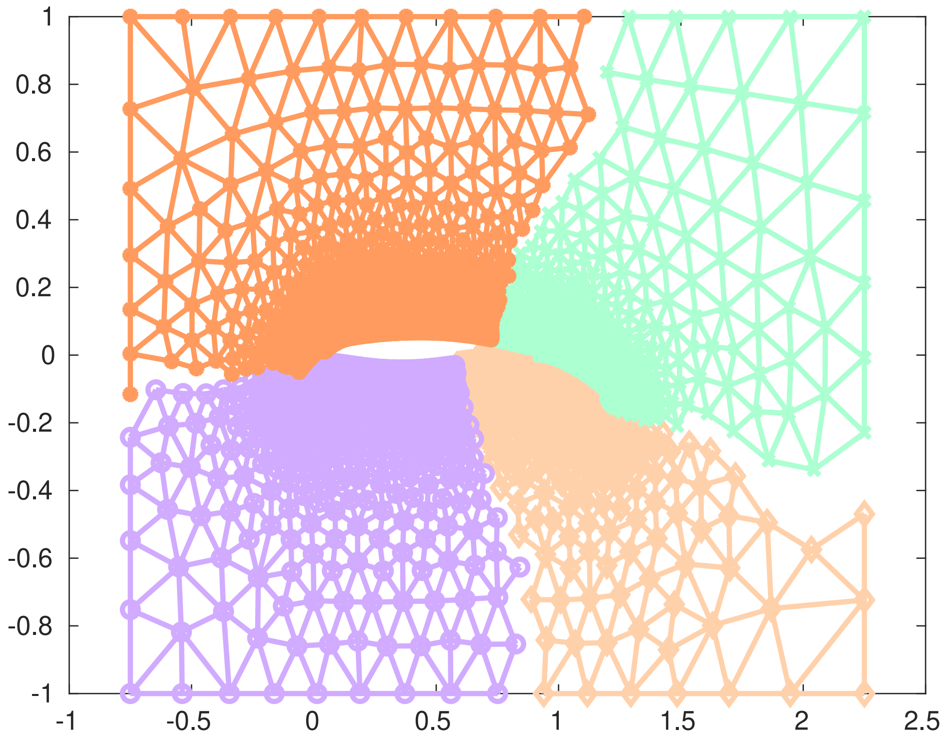
*Discretization*

↓

*Linearization (Newton)*

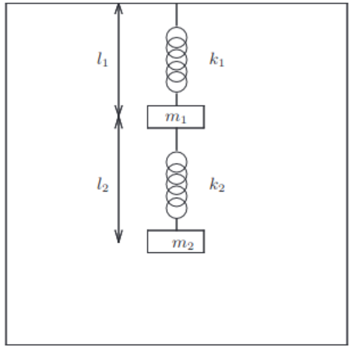
↓

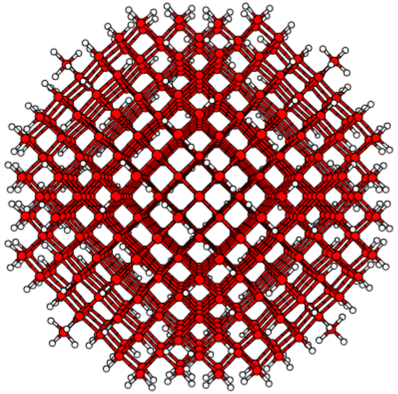
*Sparse Linear Systems* $Ax = b$

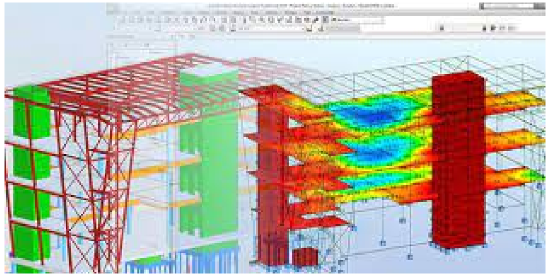


# *Example ('The old'): Eigenvalue Problems*

➤ Many applications require the computation of a few eigenvalues + associated eigenvectors of a matrix $A$



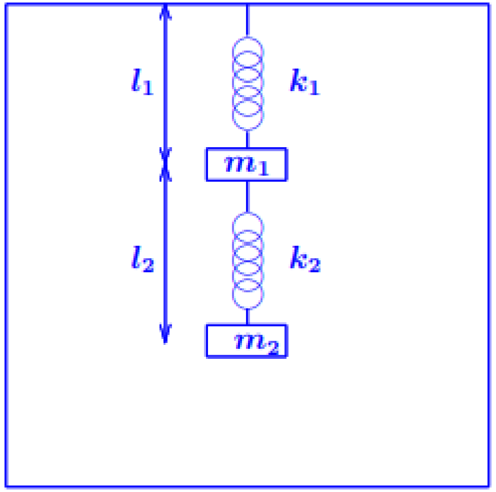





- Structural Engineering – (Goal: frequency response)
- Electronic structure calculations [Schrödinger equation..] – Quantum chemistry
- Stability analysis [e.g., electrical networks, mechanical system,..]
- ...

# *Example ('The old'): Vibrations*

➤ Vibrations in mechanical systems. See:
www.cs.umn.edu/~saad/eig_book_2ndEd.pdf

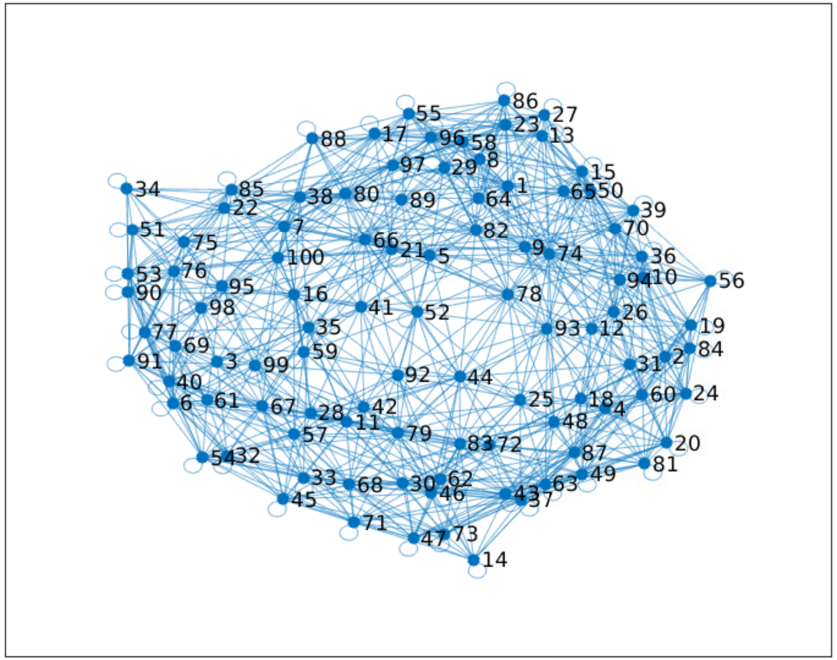



Problem: Determine the vibration modes of the mechanical system [to avoid resonance]. See details in Chapter 10 (sec. 10.2) of above reference.

➤ Problem type: Eigenvalue Problem

# *Example ('The new'): Google Rank (pagerank)*

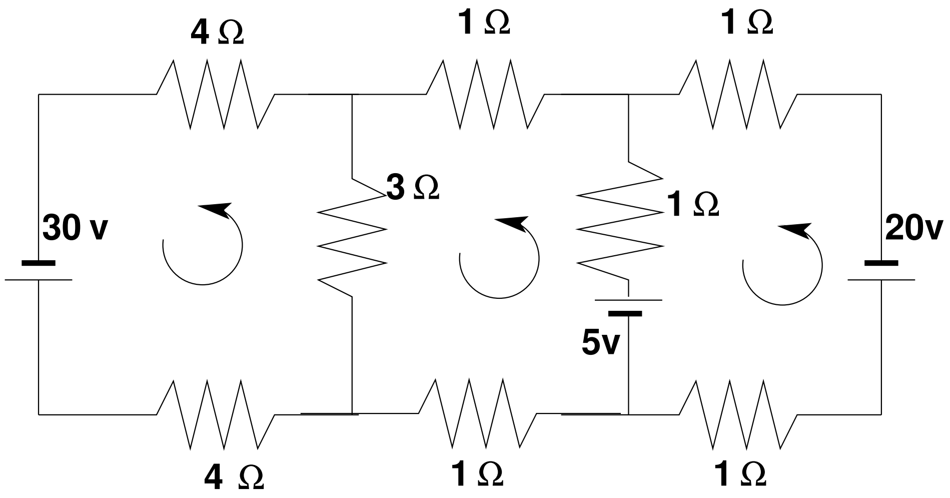
If one were to do a random walk from web page to web page, following each link on a given web page at random with equal likelihood, which are the pages to be encountered this way most often?




➤ Problem type: (homogeneous) Linear system. Eigenvector problem.

# *Example ('The old'): Power networks*

➤ Electrical circuits .. [Kirchhiff's voltage Law]

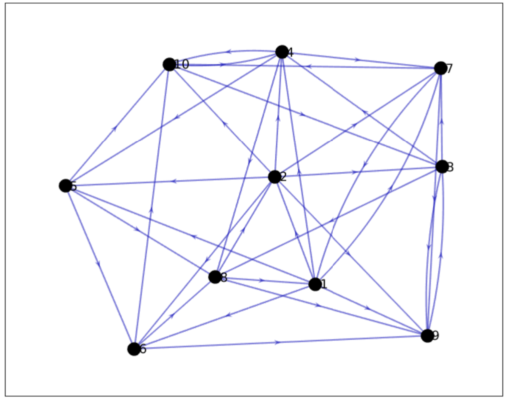



Problem: Determine the loop currents in a an electrical circuit - using Kirchhoff's Law ($V = RI$)

➤ Problem: Sparse Linear Systems [at the origin Sparse Direct Methods]

# *Example ('The new'): Economics/ Marketing/ Social Networks*

- Given: an influence graph $G$: $g_{ij}$ = strength of influence of $j$ over $i$
- Goal: charge member $i$ price $p_i$ in order to maximize profit
- Utility for member $i$: [$x_i$ = consumption of $i$]



$$u_i = ax_i - bx_i^2 + \sum_{j \neq i} g_{ij} x_j - p_i x_i$$

- 1: 'Monopolist' fixes prices; 2: agent $i$ fixes consumption $x_i$

*Result*: Optimal pricing proportional to Bonacich centrality: $(I - \alpha G)^{-1} \mathbb{1}$ where $\alpha = \frac{1}{2b}$ [*Candogan et al., 2012* + many refs.]

- ’centrality’ defines a measure of importance of a node (or an edge) in a graph
- Many other ideas of centrality in graphs [degree centrality, betweenness centrality, closeness centrality,
- Important application: Social Network Analysis

# *'Classical' Problems in Numerical Linear Algebra*

- Linear systems: $Ax = b$. Often: $A$ is large and sparse
- Least-squares problems $\min \|b - Ax\|_2$
- Eigenvalue problem $Ax = \lambda x$. Several variations -
- SVD and Low-rank approximation
- Tensors and low-rank tensor approximation
- Matrix equations: Sylvester, Lyapunov, Riccati, ..
- Nonlinear equations – acceleration methods
- Matrix functions and applications
- Many many more ...

# *'Modern' Problems in Numerical Linear Algebra*

Many of the new problems are related to datascience. A few examples:

- Low-rank approximation;
- QR; Rank-revealing QR; Updatding/Downdating QR
- Statistical methods: e.g., approximating functions of matrices
- Graph methods, Embeddings
- Network analysis, centrality
- Mixed precision linear algebra
- Fast methods based on randomization
- ...

# BACKGROUND: SOLUTION OF EIGENVALUE PROBLEMS

# *Origins of Eigenvalue Problems*

- Structural Engineering [$Ku = \lambda Mu$] (Goal: frequency response)
- Electronic structure calculations [Schrödinger equation..]
- Stability analysis [e.g., electrical networks, mechanical system,..]
- Bifurcation analysis [e.g., in fluid flow]

➤ Large eigenvalue problems in quantum chemistry use up biggest portion of the time in supercomputer centers

➤ Common problem: compute a few eigenvalues at one end of spectrum ...

➤ ... or in a given region of $\mathbb{C}$

# *Background. The Problem (s)*

- Standard eigenvalue problem:

$$Ax = \lambda x$$

Often: $A$ is symmetric real (or Hermitian complex)

- Generalized problem $Ax = \lambda Bx$ Often: $B$ is symmetric positive definite, $A$ is symmetric or nonsymmetric

- Quadratic problems: $(A + \lambda B + \lambda^2 C)u = 0$

- Nonlinear eigenvalue problems (NEVP)

$$\left[A_0 + \lambda B_0 + \sum_{i=1}^{n} f_i(\lambda) A_i\right] u = 0$$

- General form of NEVP $A(\lambda)x = 0$
- Nonlinear eigenvector problems:

$$[A + \lambda B + F(u_1, u_2, \cdots, u_k)]u = 0$$

**What to compute:**

- A few $\lambda_i$ 's with smallest or largest real parts;
- All $\lambda_i$'s in a certain region of $\mathbb{C}$;
- A few of the dominant eigenvalues;
- All $\lambda_i$'s (rare).

# *Large eigenvalue problems in applications*

➤ Some applications require the computation of a large number of eigenvalues and vectors of very large matrices.

➤ Density Functional Theory in electronic structure calculations: *'ground states'*

➤ *Excited states* involve transitions and invariably lead to much more complex computations. $\rightarrow$ Large matrices, *many* eigen-pairs to compute

## *Background: The main tools*

*Projection process:* Rayleigh-Ritz

(a) Build a 'good' subspace $\boldsymbol{K} = \mathbf{span}(\boldsymbol{V})$;

(b) get approximate eigenpairs by a Rayleigh-Ritz process:

Find $\tilde{\boldsymbol{\lambda}} \in \mathbb{C}$, $\tilde{\boldsymbol{u}} \in \boldsymbol{K}$ such that: $(\boldsymbol{A} - \tilde{\boldsymbol{\lambda}}\boldsymbol{I})\tilde{\boldsymbol{u}} \perp \boldsymbol{K}$

➤ Will revisit this shortly

## *The main tools: Shift-and-invert:*

- If we want eigenvalues near $\sigma$, replace $A$ by $(A - \sigma I)^{-1}$.

*Example:* power method: $v_j = Av_{j-1}$/scaling replaced by

$$v_j = \frac{(A-\sigma I)^{-1} v_{j-1}}{\text{scaling}}$$

- Works well for computing *a few* eigenvalues near $\sigma$/
- Used in commercial package NASTRAN (for decades!)
- Requires factoring $(A - \sigma I)$ (or $(A - \sigma B)$ in generalized case.) But convergence will be much faster.
- A solve each time - Factorization done once (ideally).

# The main tools: Deflation / Restarting

*Deflation:* ➤ Once eigenvectors converge remove them from the picture (e.g., with power method, second largest becomes largest eigenvalue after deflation).

*Restarting Strategies:*

➤ Restart projection process by using information gathered in previous steps

---

➤ ALL available methods use some combination of these ingredients.

[e.g. ARPACK: Arnoldi/Lanczos + 'implicit restarts' + shift-and-invert (option).]

# *Current state-of-the art in eigensolvers*

➤ Eigenvalues at one end of the spectrum:

- Subspace iteration + filtering [e.g. FEAST, Cheb,...]
- Lanczos+variants (thick restart, implicit restart, Davidson, filtering,..), e.g., ARPACK code, PRIMME, EVSL.
- Block Algorithms [Block Lanczos, TraceMin, LOBPCG, SlepSc,...]
- + Many others - more or less related to above

➤ 'Interior' eigenvalue problems (middle of spectrum):

- Combine shift-and-invert + Lanczos/block Lanczos. Used in, e.g., NASTRAN
- Rational filtering [FEAST, Sakurai et al.,.. ]

# THE SVD

# Background: The SVD

➤ Machine learning problems often require a (partial) *Singular Value Decomposition* -

➤ Somewhat different issues from eigenvalue problems:

- Very large matrices, update the SVD
- Compute dominant singular values/vectors
- Many problems of approximating a matrix (or a tensor) by one of lower rank (Dimension reduction, ...)

➤ But: Methods for computing SVD often based on those for standard eigenvalue problems

## *The Singular Value Decomposition (SVD)*

For any real $n \times m$ matrix $A$ there exists orthogonal matrices $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{m\times m}$ such that

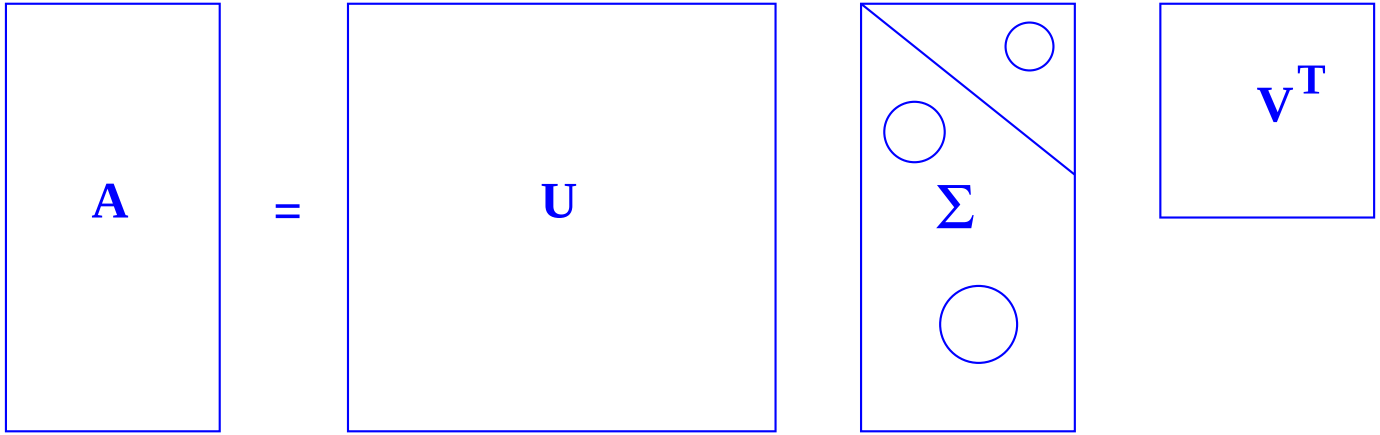
$$A = U\Sigma V^T$$

where $\Sigma$ is a diagonal matrix with nonnegative diagonal entries.

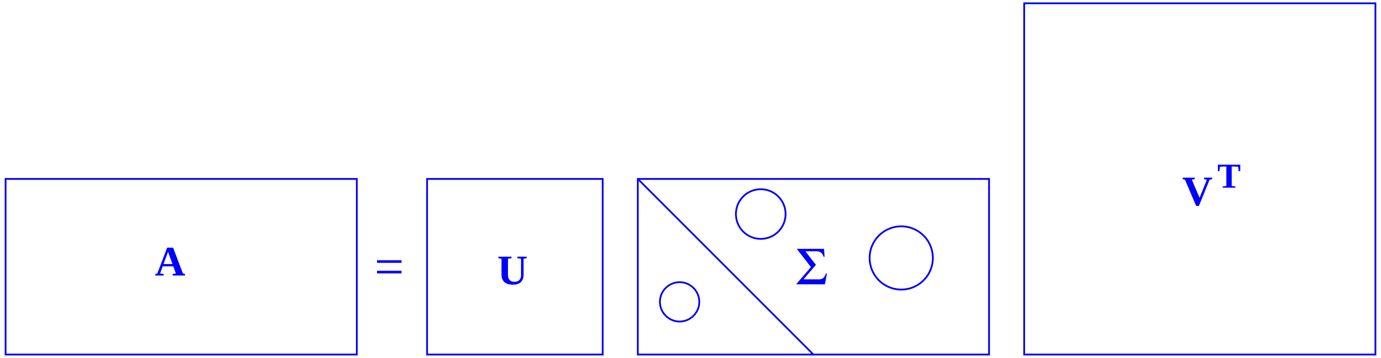
$$\sigma_{11} \geq \sigma_{22} \geq \cdots \sigma_{pp} \geq 0 \text{ with } p = \min(m, n)$$

- The $\sigma_{ii}$ are called singular values of $A$. Denoted simply by $\sigma_i$.

Case 1:




Case 2:



# The "thin" SVD

- Consider Case-1. It can be rewritten as

$$A = [U_1 U_2] \begin{bmatrix} \Sigma_1 \\ 0 \end{bmatrix} V^T \qquad \longrightarrow \qquad \boxed{A = U_1 \Sigma_1 V^T}$$

Now $U_1$ is $n \times m$ (same shape as $A$), and $\Sigma_1$ and $V$ are $m \times m$

- referred to as the "thin" SVD. Important in practice.
- Similar definition for Case 2 ['get rid of the zero block']

## Some properties. Assume that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0 \text{ and } \sigma_{r+1} = \cdots = \sigma_p = 0$$

Then:

- $rank(A) = r$ = number of nonzero singular values.
- $Ran(A) = span\{u_1, u_2, \ldots, u_r\}$
- $Null(A) = span\{v_{r+1}, v_{r+2}, \ldots, v_m\}$
- The matrix $A$ admits the SVD expansion: $$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

# Properties of the SVD (continued)

- $\|A\|_2 = \sigma_1$ = largest singular value
- $\|A\|_F = \left(\sum_{i=1}^{r} \sigma_i^2\right)^{1/2}$

➤ More generally: Schatten $p$-norm ($p \geq 1$) defined by

$$\|A\|_{*,p} = \left[\sum_{i=1}^{r} \sigma_i^p\right]^{1/p}$$

➤ Note: $\|A\|_{*,p}$ = $p$-norm of vector $[\sigma_1; \sigma_2; \cdots; \sigma_r]$

➤ In particular: $\|A\|_{*,1} = \sum \sigma_i$ is called the nuclear norm and is denoted by $\|A\|_*$. (Common in machine learning).

# Ekart-Young-Mirsky Theorem:

Let $k \le r$ and

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

then

$$\min_{rank(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

# SPARSE MATRICES ; DATA STRUCTURES

# *What are sparse matrices?*

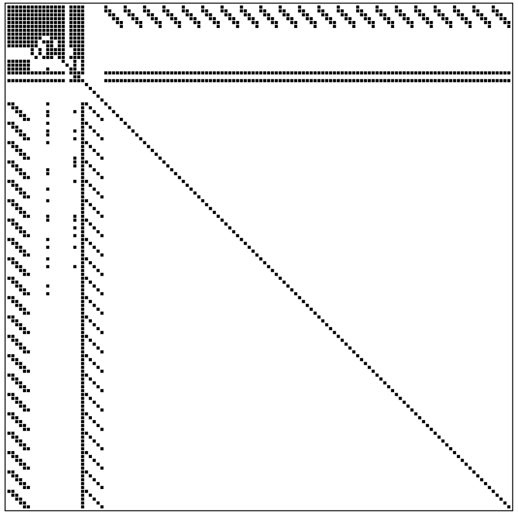
Vague definition: "..matrices that allow special techniques to take advantage of the large number of zero elements and the structure."

A few applications of sparse matrices: Structural Engineering, Reservoir simulation, Electrical Networks, optimization problems, ...

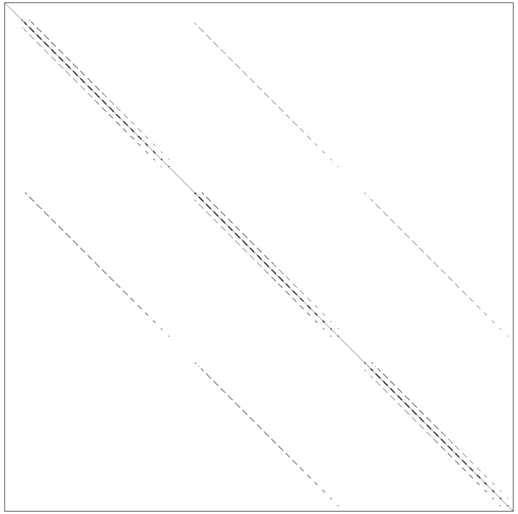
**Goals:** Much less storage and work than dense computations.

**Observation:** $A^{-1}$ is usually dense, but $L$ and $U$ in the LU factorization may be reasonably sparse (if a good technique is used).

# Sample sparsity patterns



ARC130: Unsymmetric matrix from laser problem. a.r.curtis, oct 1974



SHERMAN5: fully implicit black oil simulator 16 by 23 by 3 grid, 3 unk

## *Sparse matrices in Matlab and Python*

✍0 Explore the scripts `Lap2D`, `mark` (provided in matlab suite) for generating sparse matrices

✍1 Explore the command `spy`

✍2 Explore the command `sparse`

✍3 Run the demos titled `demo_sparse0` and `demo_sparse1`

✍4 Load the matrix `can_256.mat` from the SuiteSparse collection. Show its pattern

## *Sparse matrices - continued*

- *Main goal of Sparse Matrix Techniques:* To perform standard matrix computations economically, i.e., without storing the zeros

- *Example:* To add two square dense matrices of size $n$ requires $O(n^2)$ operations. To add two sparse matrices $A$ and $B$ requires $O(nnz(A) + nnz(B))$ where $nnz(X)$ = number of nonzero elements of a matrix $X$.

- For typical Finite Element /Finite difference matrices, number of nonzero elements is $O(n)$.

# *Data structures: The coordinate format (COO)*

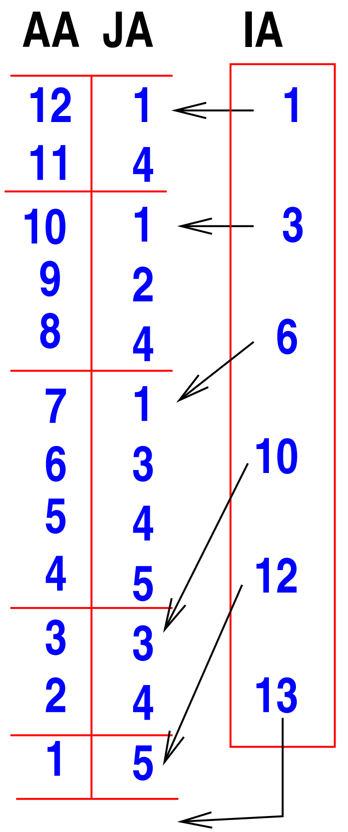
$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

| AA | JR | JC |
|---|---|---|
| 12. | 5 | 5 |
| 9. | 3 | 5 |
| 7. | 3 | 3 |
| 5. | 2 | 4 |
| 1. | 1 | 1 |
| 2. | 1 | 4 |
| 11. | 4 | 4 |
| 3. | 2 | 1 |
| 6. | 3 | 1 |
| 4. | 2 | 2 |
| 8. | 3 | 4 |
| 10. | 4 | 3 |

- Also known as ‘triplet format’
- Simple data structure - Often used as ’entry’ format in packages
- Variant used in matlab
- Note: order of entries is arbitrary [in matlab: sorted by columns]

# *Compressed Sparse Row (CSR) format*

$$A = \begin{pmatrix} 12. & 0. & 0. & 11. & 0. \\ 10. & 9. & 0. & 8. & 0. \\ 7. & 0. & 6. & 5. & 4. \\ 0. & 0. & 3. & 2. & 0. \\ 0. & 0. & 0. & 0. & 1. \end{pmatrix}$$




- IA(j) points to beginning or row j in arrays AA, JA
- Related: Compressed Sparse Column format, Modified Sparse Row format (MSR).
- Used predominantly in Fortran & portable codes [e.g. Metis] – what about C?

# CSR (CSC) format - C-style

* CSR: Collection of pointers of rows & array of row lengths

```
typedef struct SpaFmt {
/*---------------------------------------------
| C-style CSR format - used internally
| for all matrices in CSR/CSC format
|---------------------------------------------*/
  int n;           /* size of matrix          */
  int *nzcount;    /* length of each row      */
  int **ja;        /* to store column indices */
  double **ma;     /* to store nonzero entries */
} SparMat;
```

`aa[i][*]` == entries of i-th row (col.);

`ja[i][*]` == col. (row) indices,

`nzcount[i]` == number of nonzero elmts in row (col.) `i`

## Data structure used in Csparse [T. Davis' SuiteSparse code]

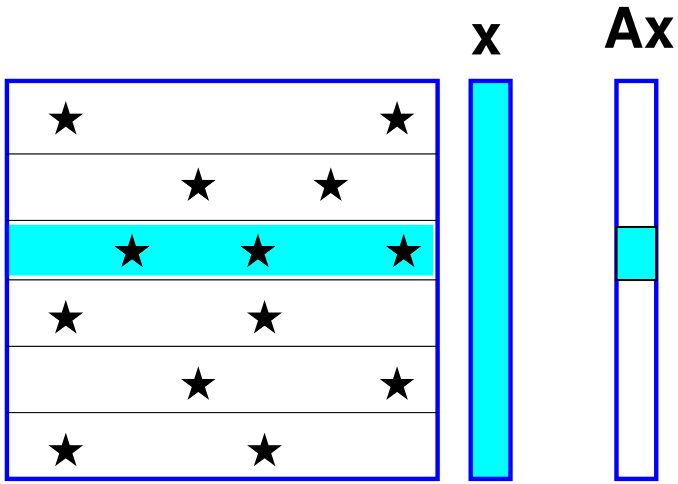
```
typedef struct cs_sparse
{/* matrix in compressed-column or triplet form */
 int nzmax ; /* maximum number of entries */
 int m ;     /* number of rows */
 int n ;     /* number of columns */
 int *p ;    /* column pointers (size n+1) or
                col indices (size nzmax) */
 int *i ;    /* row indices, size nzmax */
 double *x ; /* numerical values, size nzmax */
 int nz ;    /* # of entries in triplet matrix,
                -1 for compressed-col */
} cs ;
```

- Can be used for CSR, CSC, and COO (triplet) storage
- Easy to use from Fortran

# *Computing $y = Ax$; row and column storage*

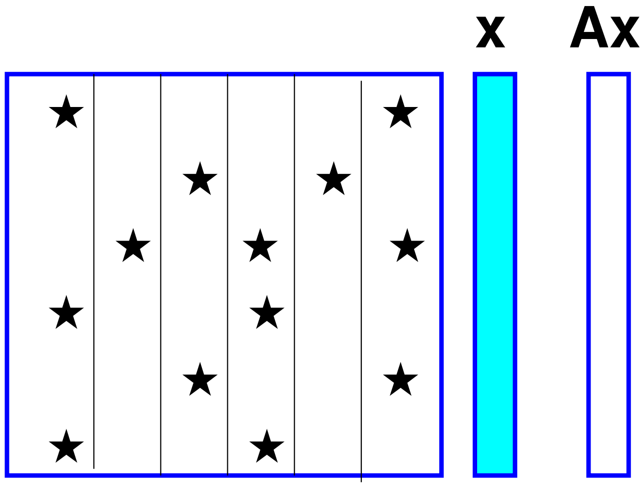
*Row-form:*

Dot product of $A(i,:)$ and $x$ gives $y_i$




*Column-form:*

Linear combination of columns $A(:,j)$ with coefficients $x_j$ yields $y$



# Matvec – row version

```
void matvec( csptr mata, double *x, double *y )
{
    int i, k, *ki;
    double *kr;
    for (i=0; i<mata->n; i++) {
        y[i] = 0.0;
        kr = mata->ma[i];
        ki = mata->ja[i];
        for (k=0; k<mata->nzcount[i]; k++)
            y[i] += kr[k] * x[ki[k]];
    }
}
```

- Uses sparse dot products (sparse SDOTS)

✍5 Operation count

## *Matvec – Column version*

```
void matvecC( csptr mata, double *x, double *y )
{
  int n = mata->n, i, k, *ki;
  double *kr;
  for (i=0; i<n; i++)
    y[i] = 0.0;
  for (i=0; i<n; i++) {
    kr = mata->ma[i];
    ki = mata->ja[i];
    for (k=0; k<mata->nzcount[i]; k++)
      y[ki[k]] += kr[k] * x[i];
  }
}
```

- Uses sparse vector combinations (sparse SAXPY)

✍6 Operation count

- Using the CS data structure from Suite-Sparse:

```
int cs_gaxpy (cs *A, double *x, double *y) {
 int p, j, n, *Ap, *Ai;
 n = A->n; Ap = A-> p; Ai = A->i; Ax = A->x;
 for (j=0; j<n; j++) {
   for (p=Ap[j]; p<Ap[j+1];p++)
     y[Ai[p]] += Ax[p]*x[j];
 }
return(1)
}
```

# Sparse matrices in matlab

✍7 Generate a tridiagonal matrix $T$

✍8 Convert $T$ to sparse format

✍9 See how you can generate this sparse matrix directly using `sparse`

✍10 See how you can use `spconvert` to achieve the same result

✍11 What can you observe about the way the triplets of a sparse matrix are ordered?

✍12 Important for performance: `spalloc`. See the difference between

`A = sparse(m,n)` and `A = spalloc(m,n,nzmax)`

✍13 Look at SparsePy for Python examples.

# BACKGROUND ON GRAPHS

## *Graphs – definitions & representations*

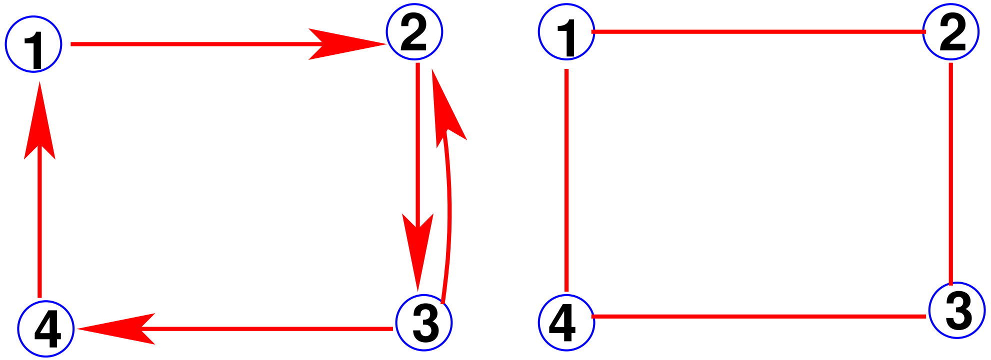
- Graph theory is a fundamental tool in many areas

**Definition.** A graph $G$ is defined as a pair of sets $G = (V, E)$ with $E \subset V \times V$. So $G$ represents a binary relation. The graph is undirected if the binary relation is symmetric. It is directed otherwise.

- $V$ is the vertex set and $E$ is the edge set
- A binary relation $R$ in $V$ can be represente by graph $G = (V, E)$ where:

$$(u, v) \in E \leftrightarrow u \; R \; v$$

Undirected graph $\leftrightarrow$ symmetric relation




(1 R 2); (4 R 1); (2 R 3); (3 R 2); (3 R 4) | (1 R 2); (2 R 3); (3 R 4); (4 R 1)

- $|E| \le |V|^2$. For undirected graphs: $|E| \le |V|(|V|+1)/2$.
- A sparse graph is one for which $|E| \ll |V|^2$.

## *Basic Terminology & notation:*

- If $(u, v) \in E$, then $v$ is adjacent to $u$. The edge $(u, v)$ is incident to $u$ and $v$.
- If the graph is directed, then $(u, v)$ is an outgoing edge from $u$ and incoming edge to $v$
- $Adj(i) = \{j | j \text{ adjacent to } i\}$
- The degree of a vertex $v$ is the number of edges incident to $v$. Can also define the indegree and outdegree. (Sometimes self-edge $i \to i$ omitted)
- $|S|$ is the cardinality of set $S$ [so $|Adj(i)|$ == deg( $i$) ]
- A subgraph $G' = (V', E')$ of $G$ is a graph with $V' \subset V$ and $E' \subset E$.

# *Representations of Graphs*

➤ A graph is nothing but a collection of vertices (indices from $1$ to $n$), each with a set of its adjacent vertices [in effect a 'sparse matrix without values']

➤ For sparse graphs: use any of the sparse matrix storage formats - omit the real values arrays.

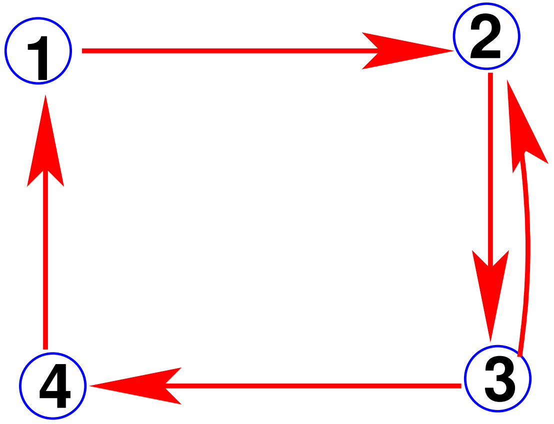
*Adjacency matrix* Assume $V = \{1, 2, \cdots, n\}$. Then the adjacency matrix of $G = (V, E)$ is the $n \times n$ matrix, with entries:

$$a_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{Otherwise} \end{cases}$$

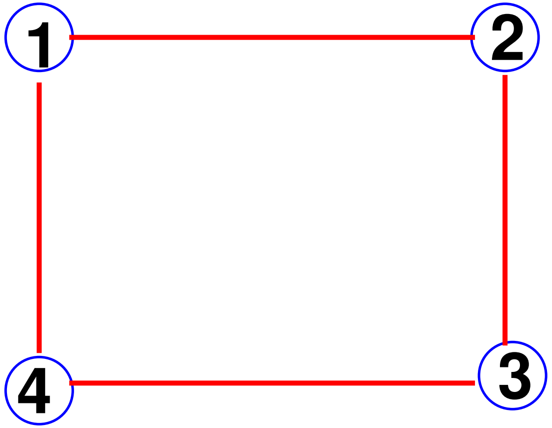
# *Representations of Graphs (cont.)*

*Example:*

$$\begin{bmatrix} & 1 & & \\ & & 1 & \\ & 1 & & 1 \\ 1 & & & \end{bmatrix}$$



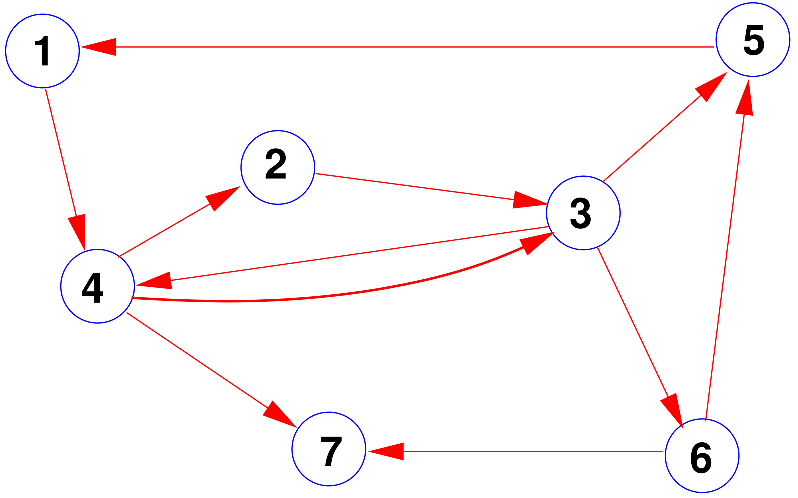

$$\begin{bmatrix} & 1 & & 1 \\ 1 & & 1 & \\ & 1 & & 1 \\ 1 & & 1 & \end{bmatrix}$$



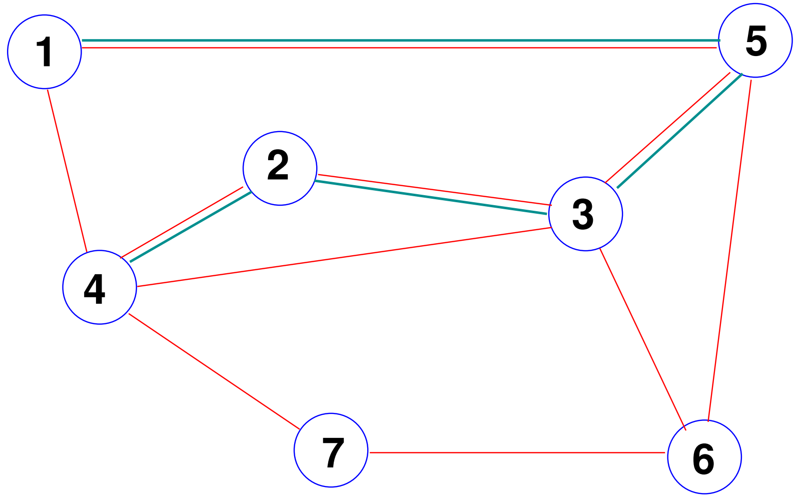
## More terminology & notation

- Given $Y \subset X$, the section graph of $Y$ is the subgraph $G_Y = (Y, E(Y))$ where $E(Y) = \{(x, y) \; \in \; E | x \; \in \; Y, \quad y \;\; in \, Y\}$
- A section graph is a clique if all the nodes in the subgraph are pairwise adjacent ($\rightarrow$ dense block in matrix)
- A path is a sequence of vertices $w_0, w_1, \ldots, w_k$ such that $(w_i, w_{i+1}) \; \in E$ for $i = 0, \ldots, k - 1$.
- The length of the path $w_0, w_1, \ldots, w_k$ is $k$ (# of edges in the path)
- A cycle is a closed path, i.e., a path with $w_k = w_0$.
- A graph is acyclic if it has no cycles.

✍14 Find cycles in this graph:




A path in an indirected graph




- A path $w_0, \ldots, w_k$ is simple if the vertices $w_0, \ldots, w_k$ are distinct (except that we may have $w_0 = w_k$ for cycles).
- An undirected graph is connected if there is path from every vertex to every other vertex.
- A digraph with the same property is said to be strongly connected

- The undirected (or symmetrized) form of a digraph = undirected graph obtained by removing the directions of all edges

- A directed graph whose undirected form is connected is said to be weakly connected or connected.

- Tree = a graph whose undirected form, i.e., symmetrized form, is acyclic & connected – Forest = a collection of trees

# Topological Sorting

*The Problem:* Given a Directed Acyclic Graph (DAG), order the vertices from $1$ to $n$ such that, if $(u, v)$ is an edge, then $u$ appears before $v$ in the ordering.

- Equivalently, label vertices from 1 to $n$ so that in any (directed) path from a node labelled $k$, all vertices in the path have labels $> k$.

Many Applications:

- Prerequisite requirements in a program
- Scheduling of tasks for any project
- Parallel algorithms;
- ...

# *Topological Sorting: A first algorithm*

Property exploited: An acyclic Digraph must have at least one vertex with indegree = 0. ✍15 Prove this

**Algorithm:**

- First label vertices with indegree 0 as as 1, 2, . . . , $k$;
- Remove these vertices and all edges incident from them
- Resulting graph is again acyclic ... $\exists$ nodes with indegree $= 0$. Label these nodes as $k+1, k+2, \ldots,$
- Repeat...

✍16 Explore implementation aspects.

➤ In practice: another algorithm is preferred: one based on Depth-First traversals of graphs.

➤ ... Details skipped

# GRAPH MODELS FOR SPARSE MATRICES

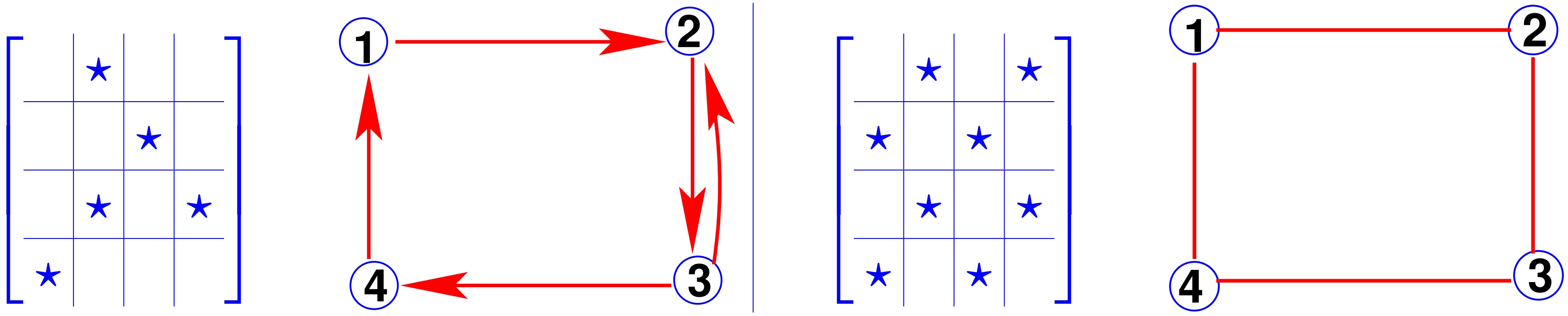
# *Graph Representations of Sparse Matrices. Recall:*

Adjacency Graph $G = (V, E)$ of an $n \times n$ matrix $A$ :

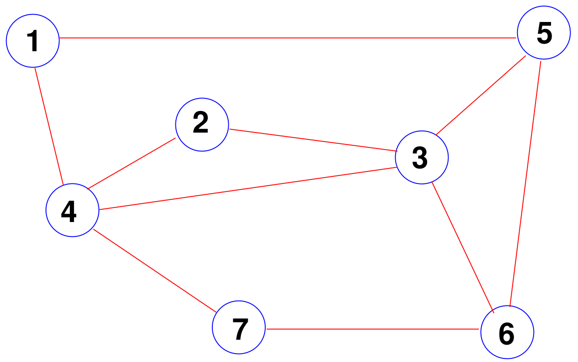
$$V = \{1, 2, ...., N\} \qquad E = \{(i, j) | a_{ij} \neq 0\}$$

- G == undirected if $A$ has a symmetric pattern

*Example:*



✍17 Show the matrix pattern for the graph on the right. The set $\{v_2, v_3, v_4\}$ is a _____? Related submatrix in adj. matrix is _____?




➤ A separator is a set $Y$ of vertices such that the graph $G_{X-Y}$ is disconnected.

*Example:* $Y = \{v_3, v_4, v_5\}$ is a separator in the above figure

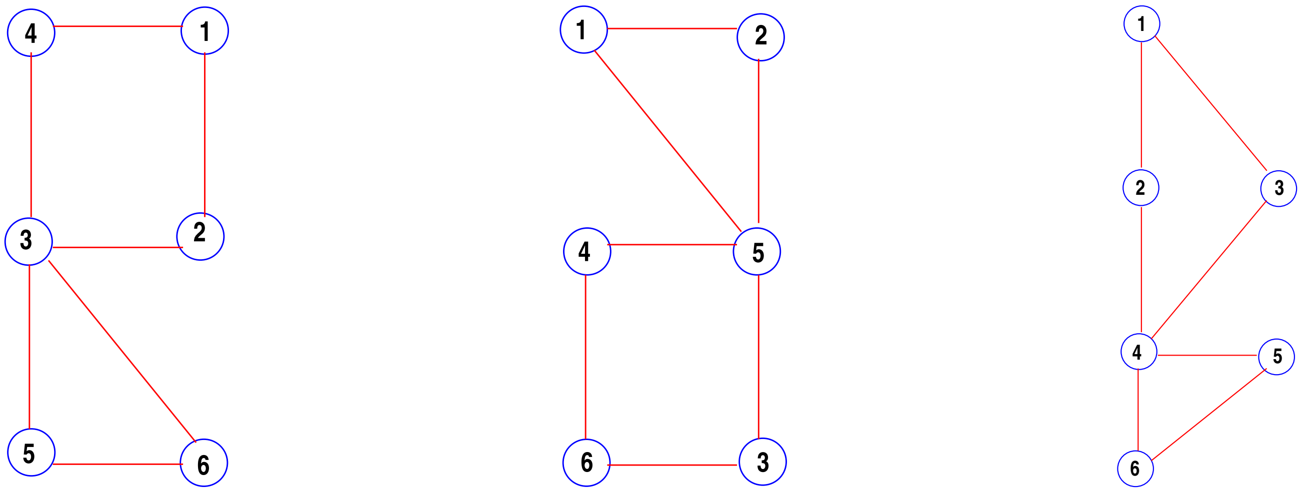
✍18 Adjacency graph of:

$$A = \begin{bmatrix} & \star & & \star & & \\ \star & & \star & & & \\ & \star & & \star & \star & \star \\ \star & & \star & & & \\ & & \star & & & \star \\ & & \star & & \star & \end{bmatrix}.$$

✍19 For any adjacency matrix $A$, what is the graph of $A^2$? [interpret in terms of paths in the graph of $A$]

- Two graphs are isomorphic is there is a mapping between the vertices of the two graphs that preserves adjacency.

✍20 Are the following 3 graphs isomorphic? If yes find the mappings between them.

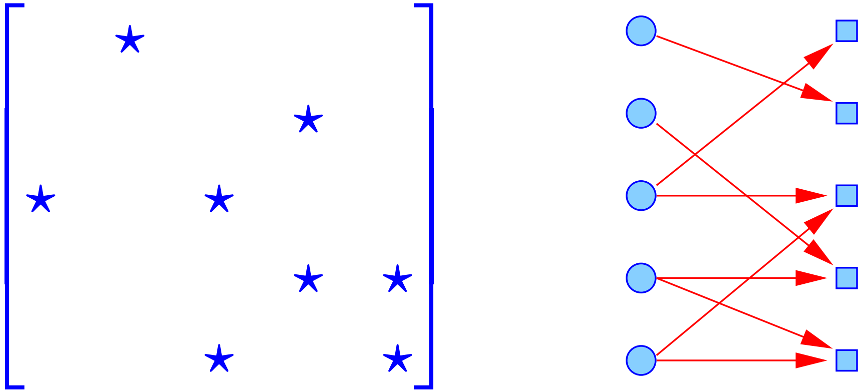



- Graphs are identical – labels are different
- Determinig graph isomorphism is a hard problem

# *Bipartite graph representation*

- Rows and columns are (both) represented by vertices;
- Relations only between rows and columns: Row $i$ is connected to column $j$ if $a_{ij} \neq 0$

*Example:*



- Bipartite models used only for specific cases [e.g. rectangular matrices, ...] - By default we use the standard definition of graphs.

## *Interpretation of graphs of matrices*

✍21 What is the graph of $A + B$ (for two $n \times n$ matrices)?

✍22 What is the graph of $A^T$ ?

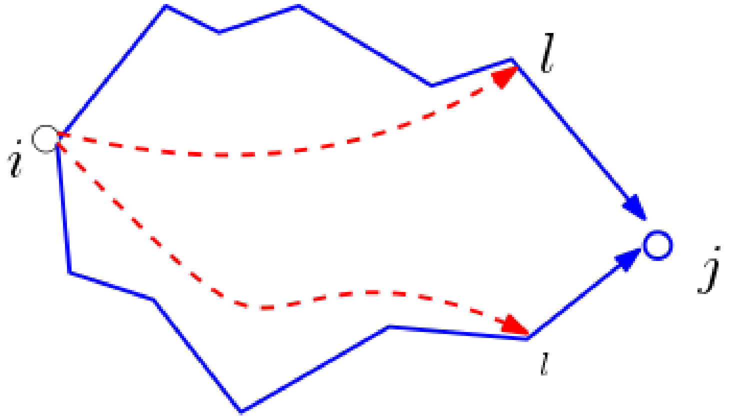
✍23 What is the graph of $A.B$?

# *Paths in graphs*

✍24 What is the graph of $A^k$?

*Theorem* Let $A$ be the adjacency matrix of a graph $G = (V, E)$. Then for $k \geq 0$ and vertices $u$ and $v$ of $G$, the number of paths of length $k$ starting at $u$ and ending at $v$ is equal to $(A^k)_{u,v}$.

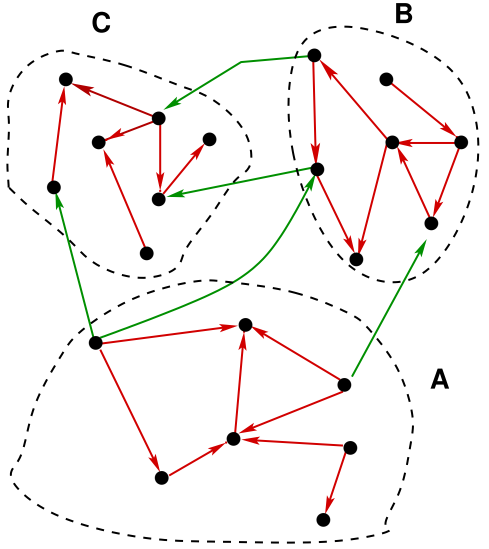
*Proof:* Proof is by induction.■

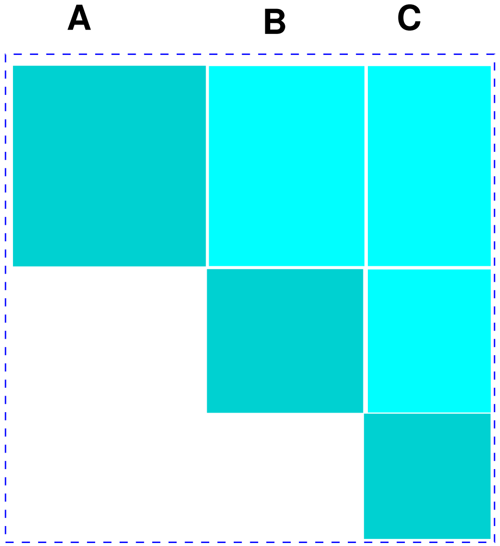



**If** $C = BA$ **then** $c_{ij} = \sum_l b_{il} a_{lj}$**. Take** $B = A^{k-1}$ **and use induction. Any path of length** $k$ **is formed as a path of length** $k-1$ **to some node** $l$ **completed by an edge from** $l$ **to** $j$**. Because** $a_{lj}$ **is one for that last edge,** $c_{ij}$ **is just the sum of all possible paths of length** $k$ **from** $i$ **to** $j$

- Recall (definition): A matrix is *reducible* if it can be permuted into a block upper triangular matrix.

- Note: A matrix is reducible iff its adjacency graph is not (strongly) connected, i.e., iff it has more than one connected component.




- No edges from $C$ to $A$ or $B$. No edges from $B$ to $A$.

**Theorem: Perron-Frobenius** An irreducible, nonnegative $n \times n$ matrix $A$ has a real, positive eigenvalue $\lambda_1$ such that:

(i) $\lambda_1$ is a simple eigenvalue of A;

(ii) $\lambda_1$ admits a positive eigenvector $u_1$ ; and

(iii) $|\lambda_i| \leq \lambda_1$ for all other eigenvalues $\lambda_i$ where $i > 1$.

- The spectral radius is equal to the eigenvalue $\lambda_1$

- Definition : a graph is $d$ regular if each vertex has the same degree $d$.

Proposition: The spectral radius of a $d$ regular graph is equal to $d$.

Proof: The vector $e$ of all ones is an eigenvector of $A$ associated with the eigenvalue $\lambda = d$. In addition this eigenvalue is the largest possible (consider the infinity norm of $A$). Therefore $e$ is the Perron-Frobenius vector $u_1$. ■

## *Application: Markov Chains*

- Read about Markov Chains in Sect. 10.9 of:

  https://www-users.cs.umn.edu/~saad/eig_book_2ndEd.pdf

- Let $\pi \equiv$ row vector of stationary probabilities
- Then $\pi$ satisfies the equation $\rightarrow$ $\pi P = \pi$

- $P$ is the probabilty transition matrix and it is 'stochastic':

A matrix $P$ is said to be *stochastic* if :

(i) $p_{ij} \geq 0$ for all $i, j$

(ii) $\sum_{j=1}^{n} p_{ij} = 1$ for $i = 1, \cdots, n$

(iii) No column of $P$ is a zero column.

- Spectral radius is $\leq 1$

✍25 Why?

- Assume $P$ is irreducible. Then:

- Perron Frobenius $\rightarrow \rho(P) = 1$ is an eigenvalue and associated eigenvector has positive entries.

- Probabilities are obtained by scaling $\pi$ by its sum.

- Example: One of the 2 models used for page rank.

*Example:* A college Fraternity has 50 students at various stages of college (Freshman, Sophomore, Junior, Senior). There are 6 potential stages for the following year: Freshman, Sophomore, Junior, Senior, graduated, or left-without degree. Following table gives probability of transitions from one stage to next

| To From | Fr | So. | Ju. | Sr. | Grad | lwd |
|---|---|---|---|---|---|---|
| Fr. | .2 | 0 | 0 | 0 | 0 | 0 |
| So. | .6 | .1 | 0 | 0 | 0 | 0 |
| Ju. | 0 | .7 | .1 | 0 | 0 | 0 |
| Sr. | 0 | 0 | .8 | .1 | 0 | 0 |
| Grad | 0 | 0 | 0 | .75 | 1 | 0 |
| lwd | .2 | .2 | .1 | .15 | 0 | 1 |

✍26 What is $P$? Assume initial population is $x_0 = [10, 16, 12, 12, 0, 0]$ and do a follow the population for a few years. What is the probability that a student will graduate? What is the probability that s/he leaves without a degree?

# *A few words on hypergraphs*

- Hypergraphs are very general.. Ideas borrowed from VLSI work

- Main motivation: to better represent communication volumes when partitioning a graph. Standard models face many limitations

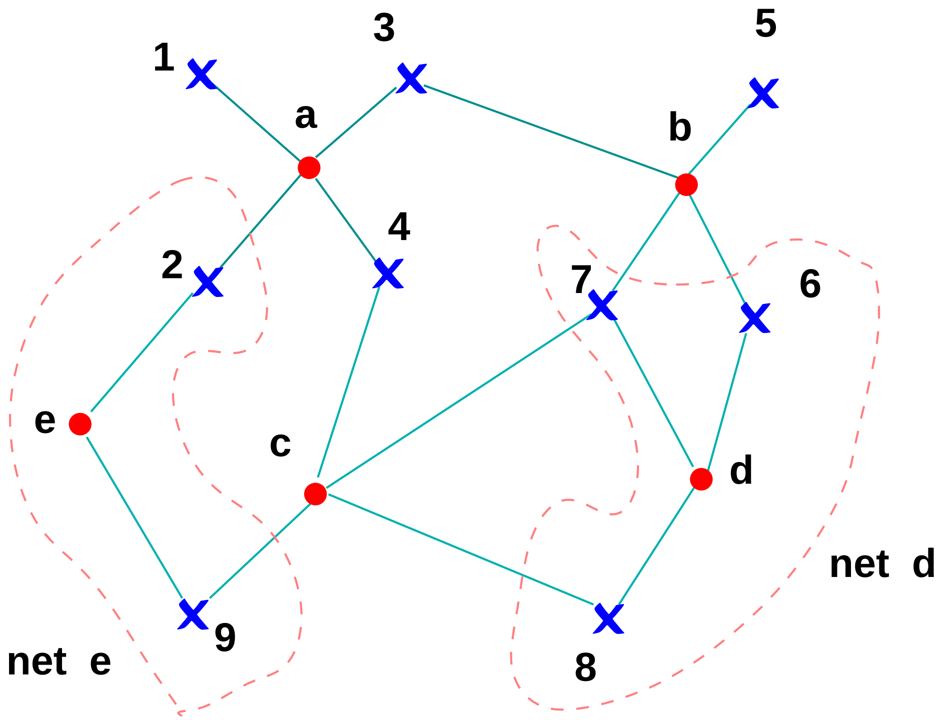
- Hypergraphs can better express complex graph partitioning problems and provide better solutions.

- Example: completely nonsymmetric patterns ...

- .. Even rectangular matrices. Best illustration: Hypergraphs are ideal for text data

**Example:** $V = \{1, \ldots, 9\}$ and $E = \{a, \ldots, e\}$ with

$a = \{1, 2, 3, 4\}$, $b = \{3, 5, 6, 7\}$, $c = \{4, 7, 8, 9\}$,
$d = \{6, 7, 8\}$, and $e = \{2, 9\}$

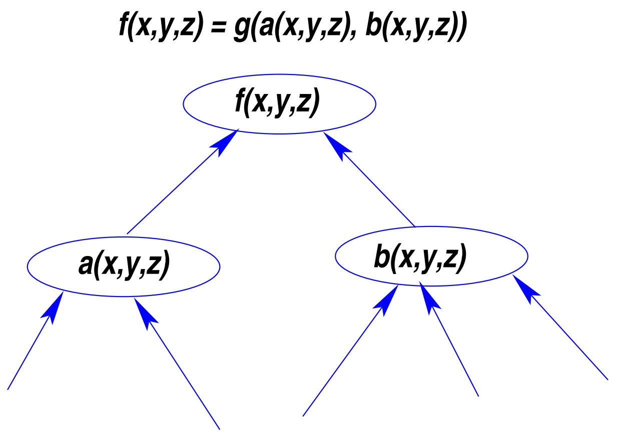



Boolean matrix:

$A =$

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | | | | | | a |
| | | 1 | | 1 | 1 | 1 | | | b |
| | | | 1 | | | 1 | 1 | 1 | c |
| | | | | | 1 | 1 | 1 | | d |
| | 1 | | | | | | | 1 | e |

# *A few words on computational graphs*




- Computational graphs: graphs where nodes represent computations whose evaluation depend on other (incoming) nodes.

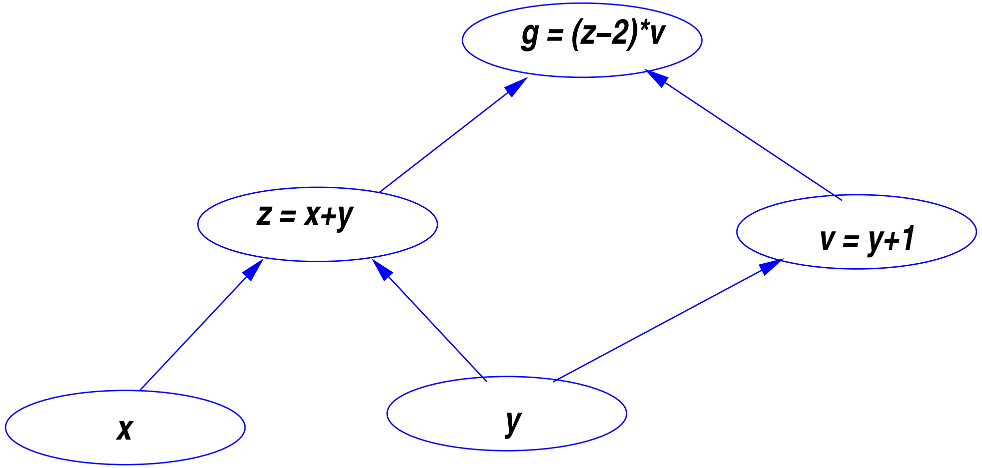
- Consider the following expression: $g(x, y) = (x + y - 2) * (y + 1)$

- Can be decomposed as: $$\begin{cases} z = x + y \\ v = y + 1 \\ g = (z - 2) * v \end{cases}$$

- Computational graph $\rightarrow$
- Given $x, y$ we want:

(a) Evaluate the nodes and

(b) derivatives w.r.t $x, y$




(a) is trivial - just follow the graph up - starting from the leaves (that contain $x$ and $y$)

(b): Use the chain rule – here shown for $x$ only using previous setting

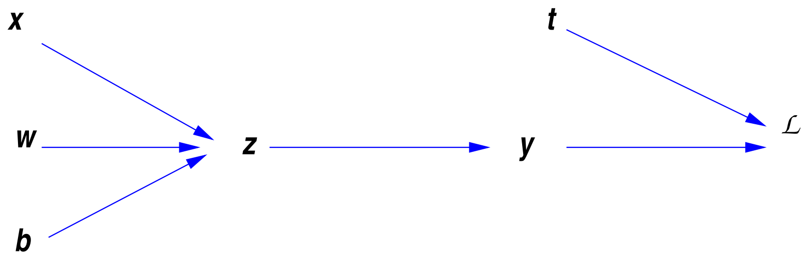
$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial a}\frac{da}{dx} + \frac{\partial g}{\partial b}\frac{db}{dx}$$

✍27 For the above example compute values and derivatives at all nodes when $x = -1, y = 2$.

# Back-Propagation

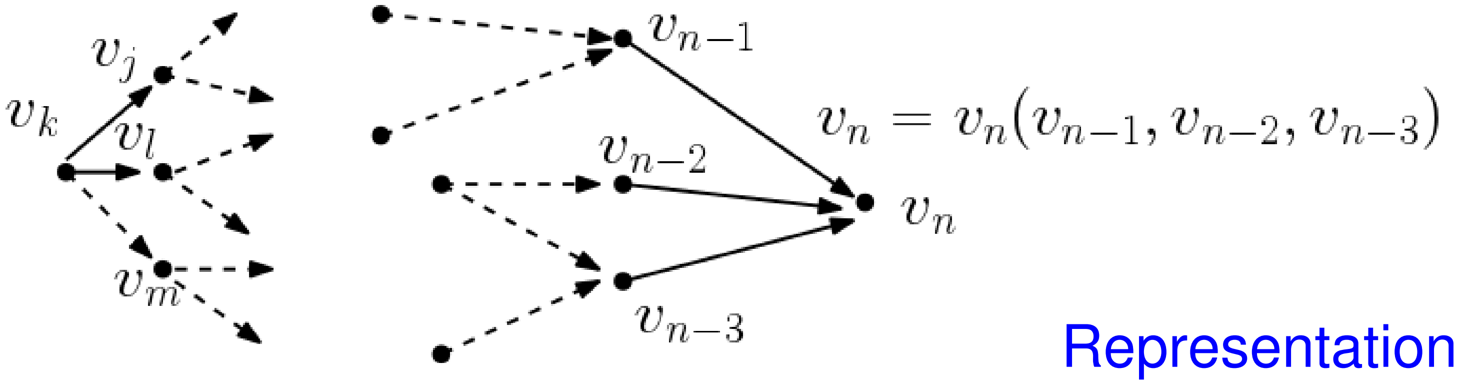
- Often we want to compute the gradient of the function at the root, once the nodes have been evaluated
- The derivatives can be calculated by going backward (or down the tree)
- Here is a very simple example from Neural Networks

$$\begin{cases} L = \frac{1}{2}(y-t)^2 \\ y = \sigma(z) \\ z = wx + b \end{cases}$$




- Note that $t$ (desired output) and $x$ (input) are constant.

# Back-Propagation: General computational graphs




Representation: *a DAG*

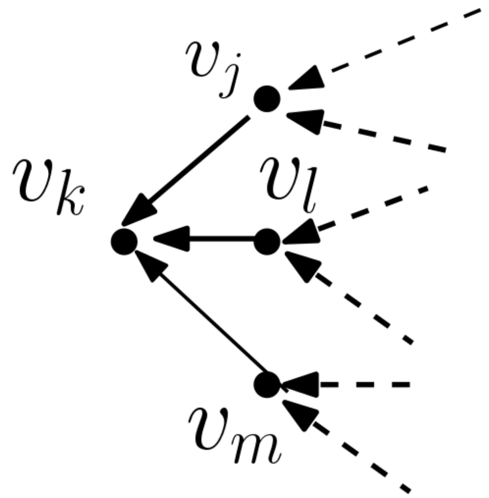
- Last node ($v_n$) is the target function. Let us rename it $f$.
- Nodes $v_i, i = 1, \cdots, e$ with indegree 0 are the variables
- Want to compute $\partial f / \partial v_1, \partial f / \partial v_2, \cdots, \partial f / \partial v_e$
- Use the chain rule. $\longrightarrow$

$$\frac{\partial f}{\partial v_k} = \frac{\partial f}{\partial v_j}\frac{\partial v_j}{\partial v_k} + \frac{\partial f}{\partial v_l}\frac{\partial v_l}{\partial v_k} + \frac{\partial f}{\partial v_m}\frac{\partial v_m}{\partial v_k}$$

- Let $\delta_k = \frac{\partial f}{\partial v_k}$ (called 'errors'). Then

$$\delta_k = \delta_j \frac{\partial v_j}{\partial v_k} + \delta_l \frac{\partial v_l}{\partial v_k} + \delta_m \frac{\partial v_m}{\partial v_k}$$

- To compute the $\delta_k$'s once the $v_j$'s have been computed (in a 'forward' propagation) – proceed backward.
- $\delta_j, \delta_l, \delta_m$ available and $\partial v_i / \partial v_k$ computable. Nore $\delta_n \equiv 1$.




- However: cannot just do this in any order. Must follow a topological order in order to obey dependencies.

- W'll revisit back-propagation later.

# GRAPH CENTRALITY

## *Centrality in graphs*

- Goal: measure importance of a node, edge, subgraph, .. in a graph
- Many measures introduced over the years
- Early Work: Freeman '77 [introduced 3 measures] – based on 'paths in graph'
- Many different ways of defininf centrality! We will just see a few

*Degree centrality:* (simplest) 'Nodes with high degree are important'

(note: scaling $n-1$ is unimportant)

$$C_D(v) = \frac{\deg(v)}{n-1}$$

*Closeness centrality:* 'Nodes that are close to many other nodes are important'

$$C_C(v) = \frac{n-1}{\sum_{w\neq v} d(v,w)}$$

*Betweenness centrality:*
(Freeman '77)

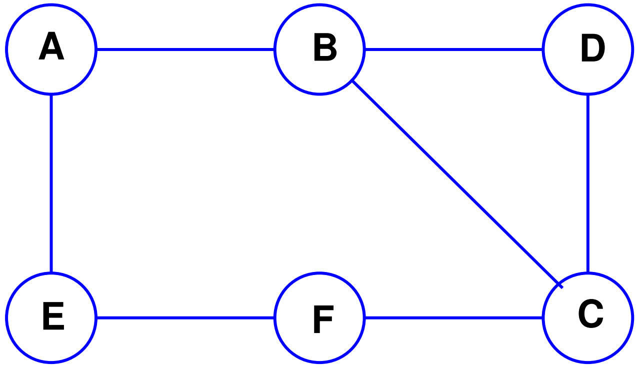
$$C_B(v) = \sum_{u\neq v, w\neq v} \frac{\sigma_{uw}(v)}{\sigma_{uw}}$$

- $\sigma_{uw}$ = total # shortest paths from $u$ to $w$
- $\sigma_{uw}(v)$ = total # shortest paths from $u$ to $w$ passing through $v$

➤ 'Nodes that are on many shortest paths are important'

✍28 Explores matlab and python codes in *centrality* folder

**Example:** Find $C_D(v)$; $C_C(v)$; $C_B(v)$ when $v = C$




| (u,w) | $\sigma_{uw}(v)$ | $\sigma_{uw}$ | / | (u,w) | $\sigma_{uw}(v)$ | $\sigma_{uw}$ | / |
|---|---|---|---|---|---|---|---|
| (A,B) | 0 | 1 | 0 | (B,E) | 0 | 1 | 0 |
| (A,D) | 0 | 1 | 0 | (B,F) | 1 | 1 | 1 |
| (A,E) | 0 | 1 | 0 | (D,E) | 1 | 2 | .5 |
| (A,F) | 0 | 1 | 0 | (D,F) | 1 | 1 | 1 |
| (B,D) | 0 | 1 | 0 | (E,F) | 0 | 1 | 0 |

- $C_D(v) = 3/5 = 0.6$ ;
- $C_C(v) = 5/[d_{CA} + d_{CB} + d_{CD} + d_{CE} + d_{CF}]$
  $= 5/[2 + 1 + 1 + 2 + 1] = 5/7$
- $C_B(v) = 2.5$ (add all ratios in table)

✍29 Redo this for $v = B$

## Eigenvector centrality:

- Supppose we have $n$ nodes $v_j$, $j = 1, \cdots, n$– each with a measure of importance ('prestige') $p_j$
- Principle: prestige of $i$ depends on that of its neighbors.
- Prestige $x_i$ = multiple of sum of prestiges of neighbors pointing to it

$$\lambda x_i = \sum_{j \in \mathcal{N}(i)} x_j = \sum_{j=1}^{n} a_{ji} x_j$$

- $x_i$ = component of eigenvector associated with $\lambda$.
- Perron Frobenius theorem at play again: take largest eigenvalue $\rightarrow$ $x_i$'s nonnegative

# *Page-rank*

➤ Can be viewed as a variant of Eigenvector centrality

**Main point:** A page is important if it is pointed to by other important pages.

➤ Importance of your page (its PageRank) is determined by summing the page ranks of all pages which point to it. [$\rightarrow$ same as EV centrality]

➤ Weighting: If a page points to several other pages, then the weighting should be distributed proportionally.

➤ Imagine many tokens doing a random walk on this graph:

- $(\delta/n)$ chance to follow one of the $n$ links on a page,
- $(1 - \delta)$ chance to jump to a random page.
- What's the chance a token will land on each page?

## Page-Rank - definitions

If $T_1$, ..., $T_n$ point to page $T_i$ then

$$\rho(T_i) \;=\; 1-\delta+\delta\left[\frac{\rho(T_1)}{|T_1|}+\frac{\rho(T_2)}{|T_2|}+\cdots\frac{\rho(T_n)}{|T_n|}\right]$$

- $|T_j|$ = count of links going out of Page $T_j$. So the 'vote' $\rho(T_j)$ is spread evenly among $|T_j|$ links.
- Sum of all PageRanks == 1: $\Sigma_T \rho(T) = 1$
- $\delta$ is a 'damping' parameter close to 1 – e.g. 0.85
- Defines a (possibly huge) Hyperlink matrix $H$

$$h_{ij} = \begin{cases} \frac{1}{|T_i|} & \text{if } i \text{ points to } j \\ 0 & \text{otherwise} \end{cases}$$

# 4 Nodes

A points to B and D

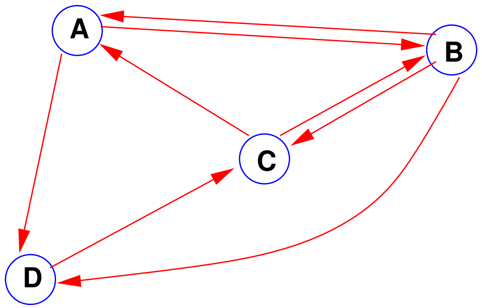
B points to A, C, and D

C points to A and B

D points to C

1) What is the H matrix?

2) the graph?




| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $A$ | | $1/2$ | | $1/2$ |
| $B$ | $1/3$ | | $1/3$ | $1/3$ |
| $C$ | $1/2$ | $1/2$ | | |
| $D$ | | | $1$ | |

- Row- sums of $H$ are = 1.
- Sum of all PageRanks will be one: $$\sum_{\text{All-Pages}_A} \rho(A) = 1.$$
- $H$ is a stochastic matrix [actually it is forced to be by changing zero rows]

**Algorithm** (PageRank)

1. Select initial row vector $v$ $(v \geq 0)$
2. For i=1:maxitr
3 $\quad v := (1 - \delta)e^T + \delta v H$
4. end

✍31 Do a few steps of this algorithm for previous example with $\delta = 0.85$.

➤ This is a row iteration..

$$v = (1 - \delta)e^T + v \,.\, \delta H$$

## A few properties:

➤ $v$ will remain $\geq 0$. [combines non-negative vectors]

More general iteration:

$$v := v\underbrace{[(1-\delta)E + \delta H]}_{G} \quad \textbf{with} \quad E = ez^T$$

where $z$ is a probability vector $e^T z = 1$ [Ex. $z = \frac{1}{n}e$]

➤ A variant of the power method.

➤ $e$ is a right-eigenvector of $G$ associated with $\lambda = 1$. We are interested in the left eigenvector.

✍32 Run *test_pr* + other drivers in /centrality

# *Kleinberg's Hubs and Authorities*

➤ Idea is to put order into the web by ranking pages by their degree of Authority or "Hubness".

➤ An Authority is a page pointed to by many important pages.

- Authority Weight = sum of Hub Weights from In-Links.

➤ A Hub is a page that points to many important pages:

- Hub Weight = sum of Authority Weights from Out-Links.

➤ Source:

http://www.cs.cornell.edu/home/kleinber/auth.pdf

# Computation of Hubs and Authorities

- Simplify computation by forcing sum of squares of weights to be 1.
- $\text{Auth}_j = x_j = \sum_{i:(i,j)\in \text{Edges}} \text{Hub}_i$.
- $\text{Hub}_i = y_i = \sum_{j:(i,j)\in \text{Edges}} \text{Auth}_j$.
- Let $A =$ Adjacency matrix: $a_{ij} = 1$ if $(i, j) \in \text{Edges}$.
- $\mathrm{y} = A\mathrm{x}$, $\mathrm{x} = A^T\mathrm{y}$.
- Iterate … to leading eigenvectors of $A^TA$ & $AA^T$.
- Answer: Leading Singular Vectors!