COC – Scientific Computing

*Degree in Computer Science*

# T2. Data Structures and Basic Operations

J. E. Roman

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Year 2021-2022



## Contents

*Section 1*

# Fields, Vectors

- Storage Formats
- Basic Operations
- Orthogonalization



## Representation of Fields: Vectors

Need to represent the different magnitudes defined in the domain (fields)

The discrete version of the field is represented associating numeric values to mesh entities:

- Nodes (vertices) - e.g. finite differences
- Cells (elements) - e.g. finite volumes
- Sometimes also in edges or faces

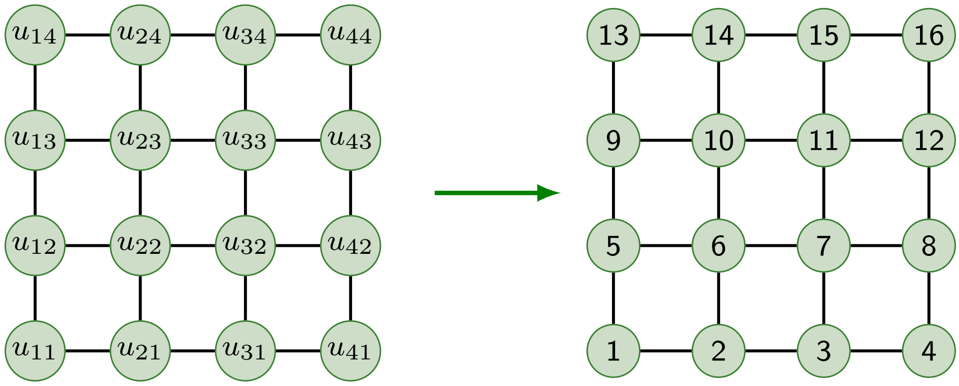
Scalar fields (e.g. pressure) are represented with 1 value, whereas vector fields (e.g. velocity) are represented with 2 or 3

It is important to keep the correspondence between elements of the vector and the associated mesh entities

## Natural Ordering

Usually the "natural ordering" is employed

- From left to right, from the bottom up




$$u = [u_{11}, u_{21}, u_{31}, u_{41}, u_{12}, \dots, u_{34}, u_{44}]^T$$

In this case, $u \in \mathbb{R}^n$ with $n = 16$

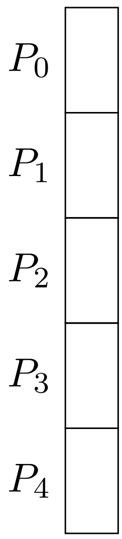
- In vector fields, $u \in \mathbb{R}^{dn}$ with $d = 2, 3$



## Parallel Vectors

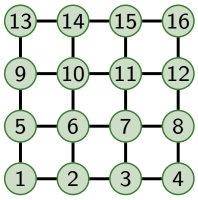
Each process locally stores a subvector

- It is convenient that the size of the subvector $n_i$ be approximately equal in all processes $i = 0, \dots, p-1$
- Local indices $1, \dots, n_i$ correspond to global indices



## Parallel Vectors: Example

We want to distribute the vector among several processes:




Block distribution, $p = 2$

$\rightarrow P_0 = \{1, 2, 3, 4, \dots, 8\}, P_1 = \{9, 10, 11, 12, \dots, 16\}$

Block distribution, $p = 3$

$\rightarrow P_0 = \{1, 2, \dots, 6\}, P_1 = \{7, 8, \dots, 11\}, P_2 = \{12, 13, \dots, 16\}$

Block cyclic distribution, $n_b = 4$, $p = 2$

$\rightarrow P_0 = \{1, 2, 3, 4, 9, 10, 11, 12\}, P_1 = \{5, 6, 7, 8, 13, 14, 15, 16\}$

Block cyclic distribution, $n_b = 4$, $p = 3$

$\rightarrow P_0 = \{1, 2, 3, 4, 13, 14, 15, 16\}, P_1 = \{5, 6, 7, 8\}, P_2 = \{9, 10, 11, 12\}$



## Vector Operations

Let $x, y, z \in \mathbb{R}^n$ and $\alpha, r \in \mathbb{R}$

Sum and product by a scalar

- Sum: $z = x + y, \quad z_i = x_i + y_i$
- Scaling: $y = \alpha x, \quad y_i = \alpha x_i$
- Combined operation (AXPY): $y = \alpha x + y$
- Trivially parallelizable

Vector dot product (also *scalar* product or *inner* product)

- $r = x^T y$

$$r = \sum_{i=1}^{n} x_i y_i$$

- In parallel: reduction of the partial sums

## Vector Norm

A vector norm is a real function $\|\cdot\|$ satisfying

1. $\|x\| \geq 0 \quad \forall x \in \mathbb{R}^n, \quad \|x\| = 0 \Leftrightarrow x = 0$
2. $\|x + y\| \leq \|x\| + \|y\| \quad \forall x, y \in \mathbb{R}^n$
3. $\|\alpha x\| = |\alpha| \|x\| \quad \forall \alpha \in \mathbb{R} \quad \forall x \in \mathbb{R}^n$

Vector 2-norm or Euclidean norm

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{x^T x}$$

Other: 1-norm, $\infty$-norm

---

Normalization: to scale a vector so that its norm is 1

$$\tilde{x} = \frac{x}{\|x\|_2}$$

One can check that $\|\tilde{x}\|_2 = 1$



## Multi-Vectors

Given $k$ vectors $\{v_1, v_2, \ldots, v_k\}$, they are sometimes combined in a matrix

$$V = \left[ \begin{array}{c|c|c|c} v_1 & v_2 & \cdots & v_k \end{array} \right] \in \mathbb{R}^{n \times k}$$

Operations

- Multiple AXPY: let $s = [s_1, s_2, \ldots, s_k]^T$

$$y = y + Vs = y + \sum_{i=1}^{k} s_i v_i$$

- Multiple dot product: $s = V^T x$ where $s_i = v_i^T x$

## Vector Subspaces

$\mathcal{V} = \mathrm{span}\{v_1, v_2, \ldots, v_k\}$ is the subspace spanned by the vectors $v_1, \ldots, v_k$

- If the $v_i$'s are linearly independent, then they form a basis of $\mathcal{V}$ and $\dim(\mathcal{V}) = k$
- Otherwise, $\dim(\mathcal{V}) < k$

Let $V = [v_1, v_2, \ldots, v_k]$ and $s = [s_1, s_2, \ldots, s_k]^T$

- $y = Vs$ is a linear combination of the $v_i$'s
- Therefore, $y \in \mathcal{V}$

Let $t = [t_1, t_2, \ldots, t_k]^T$ and $s = Ht$

- $y = Vs = VHt = Wt$
- Change of basis: the columns of $W = VH$ generate the same subspace (if $H$ is non-singular)



## Orthogonalization of Vectors

Orthogonal vectors

- Two vectors $x, y \in \mathbb{R}^n, x \neq 0, y \neq 0$ are orthogonal if $x^T y = 0$
- A set $\{q_1, q_2, \ldots, q_k\}$ is orthogonal if $q_i^T q_j = 0$ for $i \neq j$
- We say it is orthonormal if in addition $q_i^T q_i = 1$
- In matrix form: $Q^T Q = I$ ($I =$identity matrix)

---

Problem: obtain an orthonormal basis of a subspace

- Input: $V$ such that $\mathcal{V} = \mathrm{span}\{v_1, v_2, \ldots, v_k\}$
- Output: $Q$ such that the subspace generated by the $q_i$'s is also $\mathcal{V}$ (change of basis) and they are orthonormal

## Gram-Schmidt: Idea

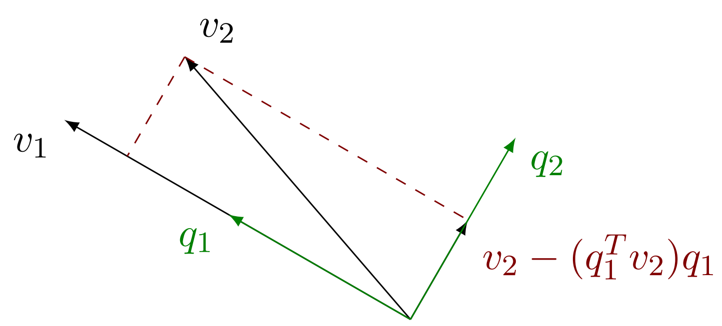
The Gram-Schmidt method generates the $q_i$'s one by one

First vector:

- Normalize: $q_1 = v_1/\|v_1\|_2$ (assuming $\|v_1\| \neq 0$)

Second vector:

- Subtract its projection on $q_1$: $\hat{v}_2 = v_2 - (q_1^T v_2)q_1$
- Normalize: $q_2 = \hat{v}_2/\|\hat{v}_2\|_2$ (assuming $\|\hat{v}_2\| \neq 0$)




In general: subtract from $v_j$ its projection on $q_1, q_2, \ldots, q_{j-1}$



## Classical Gram-Schmidt

Given a basis $\{v_1, v_2, \ldots, v_k\}$ obtain an orthonormal basis of the same subspace $\{q_1, q_2, \ldots, q_k\}$

**CGS: Classical Gram-Schmidt**

$r_{11} = \|v_1\|_2$ (if $r_{11} = 0$ abort)
$q_1 = v_1/r_{11}$
**for** $j = 2, \ldots, k$
  **for** $i = 1, \ldots, j-1$
    $r_{i,j} = q_i^T v_j$
  **end**
  $\hat{v}_j = v_j$
  **for** $i = 1, \ldots, j-1$
    $\hat{v}_j = \hat{v}_j - r_{i,j} q_i$
  **end**
  $r_{j,j} = \|\hat{v}_j\|$ (if $r_{j,j} = 0$ abort)
  $q_j = \hat{v}_j/r_{j,j}$
**end**

## Modified Gram-Schmidt

**MGS: Modified Gram-Schmidt**

$r_{11} = \|v_1\|_2$ (if $r_{11} = 0$ abort)
$q_1 = v_1 / r_{11}$
**for** $j = 2, \ldots, k$
  $\hat{v}_j = v_j$
  **for** $i = 1, \ldots, j-1$
    $r_{i,j} = q_i^T \hat{v}_j$
    $\hat{v}_j = \hat{v}_j - r_{i,j} q_i$
  **end**
  $r_{j,j} = \|\hat{v}_j\|$ (if $r_{j,j} = 0$ abort)
  $q_j = \hat{v}_j / r_{j,j}$
**end**

MGS numerically more stable than CGS
CGS more efficient in parallel: the scalar products $r_{1:j-1,j} = Q_{j-1}^T v_j$ are merged in a single communication



## QR Decomposition

The result of Gram-Schmidt satisfies $v_j = \sum_{i=1}^{j} r_{i,j} q_i$

In matrix notation: $V = QR$, being $R$ upper triangular

$$\left[\begin{array}{c|c|c|c} v_1 & v_2 & \cdots & v_k \end{array}\right] = \left[\begin{array}{c|c|c|c} q_1 & q_2 & \cdots & q_k \end{array}\right] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1,k} \\ & r_{22} & \cdots & r_{2,k} \\ & & \ddots & \vdots \\ & & & r_{k,k} \end{bmatrix}$$

$R$ is the matrix of the change of basis

*Section 2*

# Sparse Matrices

- Matrix Properties and Types
- Storage Formats
- Basic Operations



## Symmetric Matrices

A square matrix $A \in \mathbb{R}^{n\times n}$ is symmetric if $a_{ij} = a_{ji}$

$$A = A^T$$

Symmetric positive-definite matrix: satisfies

$$x^T Ax > 0 \quad \forall x \in \mathbb{R}^n,\ x \neq 0$$

Characterization:

- a matrix is SPD iff $\det(A_{1:i,1:i}) > 0, \quad 1 \leq i \leq n$

Example:

$$A = \begin{bmatrix} 5 & 2 & -1 \\ 2 & 2 & 1 \\ -1 & 1 & 3 \end{bmatrix}$$

$$\det\left(\begin{bmatrix} 5 \end{bmatrix}\right) = 5 > 0$$

$$\det\left(\begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}\right) = 6 > 0$$

$$\det(A) = 7 > 0$$

## Orthogonal Matrices

A square matrix $Q \in \mathbb{R}^{n \times n}$ is orthogonal if its columns are orthonormal

$$Q^T Q = I$$

Example: $Q = \begin{bmatrix} 1/2 & 0 & \sqrt{3}/2 \\ 0 & 1 & 0 \\ -\sqrt{3}/2 & 0 & 1/2 \end{bmatrix}$

Properties:

- $Q^{-1} = Q^T$
- $\|Qx\|_2 = \|x\|_2$

Permutation matrix: orthogonal matrix whose elements are zero except a 1 in each row and column

Example: $P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Sometimes represented as an index vector $p = [2, 3, 1, 4]$



## Diagonally Dominant Matrices

We say a matrix is diagonally dominant

by rows:
$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|, \; i = 1, \ldots, n$

by columns:
$|a_{jj}| \geq \sum_{i \neq j} |a_{ij}|, \; j = 1, \ldots, n$

In case of strict inequality, we say strictly DD
$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \; i = 1, \ldots, n$

Examples:

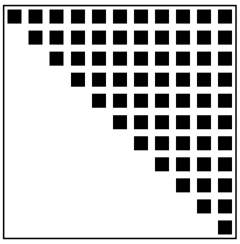
$$A = \begin{bmatrix} -3 & 0 & 0 & -1 \\ 2 & 6 & 3 & 0 \\ 2 & 1 & -6 & -2 \\ 3 & 0 & 0 & -4 \end{bmatrix}$$

Strictly DD by rows

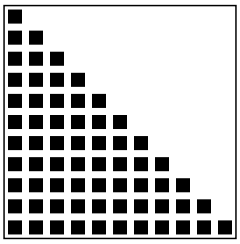
$$T = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix}$$

Diagonally dominant

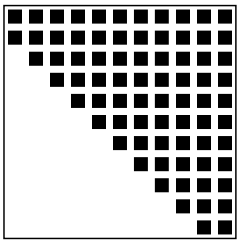
## Triangular Matrices



Upper triangular
$a_{ij} = 0$ for $i > j$



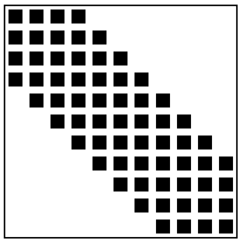
Lower triangular
$a_{ij} = 0$ for $i < j$



Upper Hessenberg
$a_{ij} = 0$ for $i > j + 1$

Band matrix
$a_{ij} = 0$ if $|i - j| > \beta$ ($\beta$ =bandwidth)

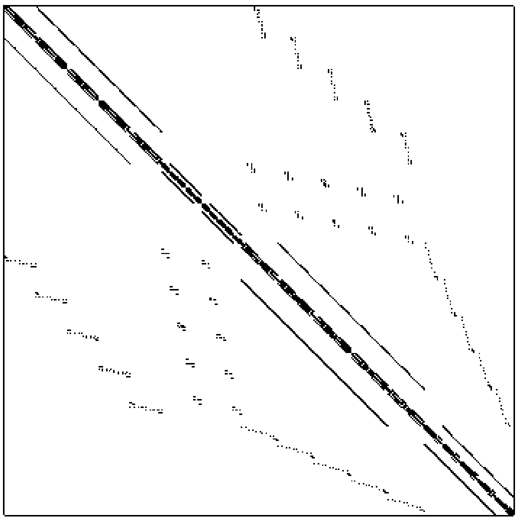
Particular cases: diagonal ($\beta$=0), tridiagonal ($\beta$=1), pentadiagonal ($\beta$=2)

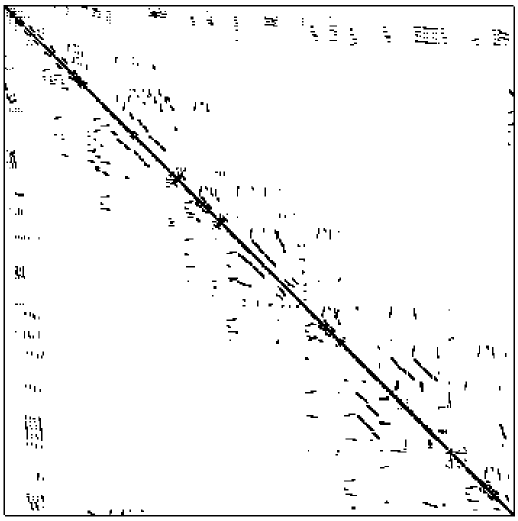




## Sparse Matrices

In mesh-based applications, the percentage of nonzero elements is usually very small (<1%)





The structure or nonzero pattern depends on

- The connectivity between unknowns (according to discretization)
- The ordering that has been chosen for the unknowns

## Sparse Storage Formats

Special data structures, to:

- Reduce the memory requirements
- Reduce the cost of operations

There are many different formats – intended to optimize the most frequent operations

- Matrix-vector product
  The most frequent one, requires maximum efficiency
- Extraction of the diagonal
- Factorization
  Produces "fill-in", new nonzero elements appear
- Other operations: addition, matrix-matrix product
- Insert/delete elements
  Infrequent, constant nonzero pattern



## Coordinate Format (COO)

Uses three arrays:

| | | |
|---|---|---|
| `A` | `nz` | numeric values |
| `I` | `nz` | row indices |
| `J` | `nz` | column indices |

(`nz` is the number of nonzero elements)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| `A` | $a_{11}$ | $a_{13}$ | $a_{21}$ | $a_{22}$ | $a_{24}$ | $a_{33}$ | $a_{44}$ | $a_{45}$ | $a_{52}$ | $a_{53}$ | $a_{55}$ |
| `I` | 1 | 1 | 2 | 2 | 2 | 3 | 4 | 4 | 5 | 5 | 5 |
| `J` | 1 | 3 | 1 | 2 | 4 | 3 | 4 | 5 | 2 | 3 | 5 |

$$\begin{bmatrix} a_{11} & & a_{13} & & \\ a_{21} & a_{22} & & a_{24} & \\ & & a_{33} & & \\ & & & a_{44} & a_{45} \\ & a_{52} & a_{53} & & a_{55} \end{bmatrix}$$

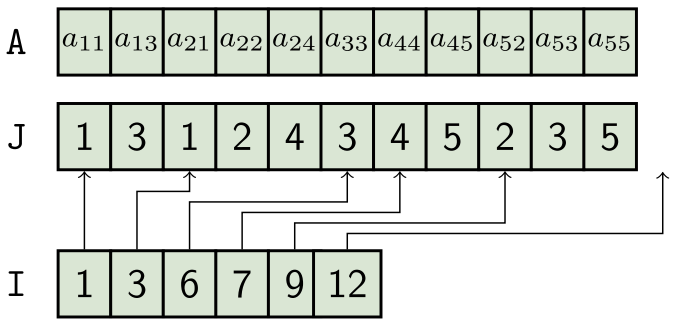
- Very efficient sequential access
- Inefficient element search

# Compressed Sparse Row (CSR)

Uses three arrays:

| | | |
|---|---|---|
| `A` | `nz` | numeric values |
| `I` | `n+1` | pointers to row start |
| `J` | `nz` | column indices |

(`n` is the number of rows of the matrix)




$$\begin{bmatrix} a_{11} & & a_{13} & & \\ a_{21} & a_{22} & & a_{24} & \\ & & a_{33} & & \\ & & & a_{44} & a_{45} \\ & a_{52} & a_{53} & & a_{55} \end{bmatrix}$$

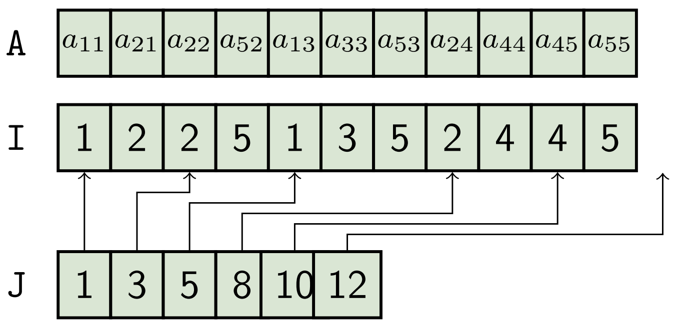
- Row indices are not stored explicitly
- `I[k]` indicates the position in which row `k` starts



# Variants

## Compressed Sparse Column (CSC)

- Analog of CSR, but sorted by columns
- `J[k]` indicates the position in which column `k` starts




$$\begin{bmatrix} a_{11} & & a_{13} & & \\ a_{21} & a_{22} & & a_{24} & \\ & & a_{33} & & \\ & & & a_{44} & a_{45} \\ & a_{52} & a_{53} & & a_{55} \end{bmatrix}$$

## Modified Sparse Row (MSR)

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `A` | $a_{11}$ | $a_{22}$ | $a_{33}$ | $a_{44}$ | $a_{55}$ | * | $a_{13}$ | $a_{21}$ | $a_{24}$ | $a_{45}$ | $a_{52}$ | $a_{53}$ |
| `IJ` | 1 | 2 | 4 | 4 | 5 | 7 | 3 | 1 | 4 | 5 | 2 | 3 |

The main diagonal is stored separately

## Sparse Matrix Operations

Let $A, B \in \mathbb{R}^{m \times n}$, $v \in \mathbb{R}^n$, $w \in \mathbb{R}^m$ and $\alpha \in \mathbb{R}$

Sum and product by a scalar

- Matrix AXPY: $B = \alpha A + B$
- Difficult if the pattern of $A$ is not a subset of $B$'s

Product of matrices, $A = CD$

- The dimensions of $C$ and $D$ must be consistent
- Very uncommon in sparse matrices

Matrix-vector product, $w = Av$

- Fundamental operation in iterative methods



## Matrix Norm

Definition is analog of vector norm

Frobenius norm: $\|A\|_F = \sqrt{\sum_{i,j} a_{i,j}^2}$

Matrix 2-Norm (also 1-norm, $\infty$-norm)

- Induced from the vector 2-norm

$$\|A\|_2 = \max_{\|y\|_2 = 1} \|Ay\|_2$$

- Intuitively: maximum produced elongation
- Consistent norms: $\|Ax\|_2 \leq \|A\|_2 \|x\|_2$

Condition number

$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2, \quad \kappa_2(A) \geq 1$$

## Matrix-Vector Product Algorithm

$w = Av$

**Matrix-vector product by rows**

$w = [0, 0, \ldots, 0]^T$
**for** $i = 1, \ldots, m$
    **for** $j = 1, \ldots, n$
        $w[i] += A[i][j] * v[j]$
    **end**
**end**

**Matrix-vector prod. COO**

$w = [0, 0, \ldots, 0]^T$
**for** $k = 1, \ldots, nz$
    $w[I[k]] += A[k] * v[J[k]]$
**end**

**Matrix-vector prod. CSR**

$w = [0, 0, \ldots, 0]^T$
**for** $i = 1, \ldots, m$
    **for** $k = I[i], \ldots, I[i+1]-1$
        $w[i] += A[k] * v[J[k]]$
    **end**
**end**



*Section 3*

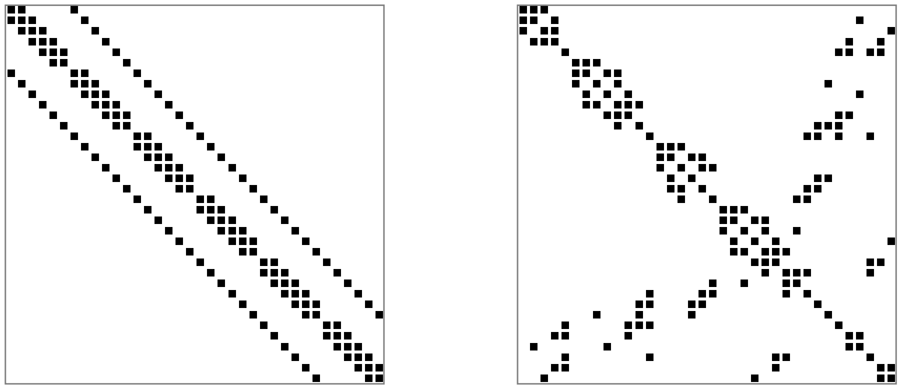
# Orderings and Partitioning

- Meshes, Graphs
- Orderings
- Partitioning

## Sparse Pattern

The nonzero element pattern depends on

- The connectivity in the mesh
- The chosen ordering



The ordering can be chosen to improve some structural property of the matrix (bandwidth, fill-in, ...)

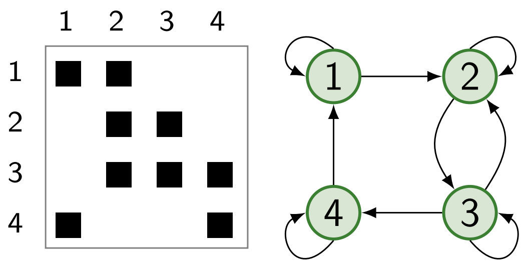


## Adjacency Graph

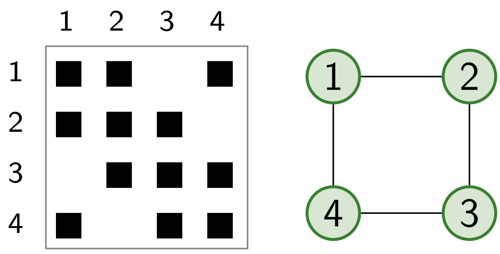
We use graph theory to study the sparsity structure

Adjacency Graph of a sparse matrix $A \in \mathbb{R}^{n \times n}$, $G(A) = (V, E)$

- The $n$ vertices of $V$ represent the unknowns
- The edges $E$ represent binary relations among them
  - An edge $(v_i, v_j) \in E$ if $a_{ij} \neq 0$
  - Represents the presence of unknown $j$ in equation $i$



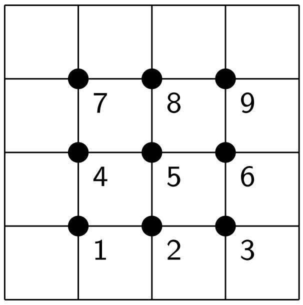

Non-symmetric; Directed graph



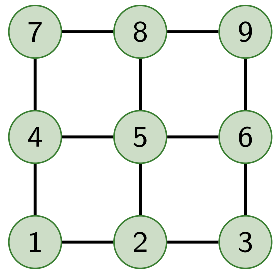

Symmetric; Undirected graph

# Correspondence between Mesh and Graph

Discretization mesh




Choose an ordering




Adjacency graph

System matrix

$$A = \begin{bmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{bmatrix}$$



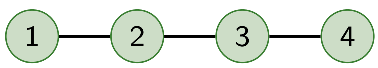
# Symmetric Permutation

Given a system $Ax = b$, and let $P$ be a permutation matrix

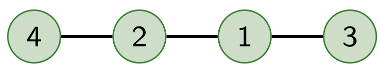
- Row permutation $PAx = Pb$, reorders equations
- Column permutation $AP^T Px = b$, exchanges unknowns
- Symmetric permutation $(PAP^T)Px = Pb$, modifies both

Example with permutation $p = [3, 2, 4, 1]$:

$$A = \begin{bmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & a_{43} & a_{44} \end{bmatrix}, \quad \begin{bmatrix} a_{33} & a_{32} & a_{34} & \\ a_{23} & a_{22} & & a_{21} \\ a_{43} & & a_{44} & \\ & a_{12} & & a_{11} \end{bmatrix} \begin{bmatrix} x_3 \\ x_2 \\ x_4 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_3 \\ b_2 \\ b_4 \\ b_1 \end{bmatrix}$$




Adjacency graph

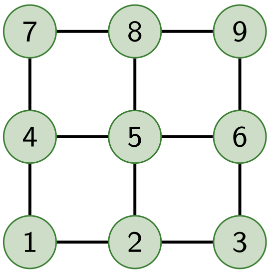



Graph after permutation

$\rightarrow$ the graph vertices are relabelled without modifying edges

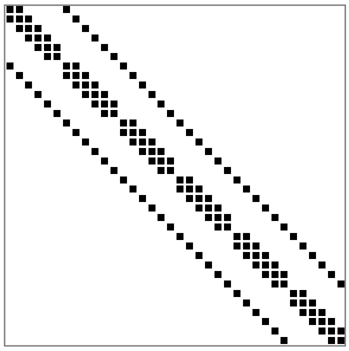
# Red-Black Ordering

Natural Ordering

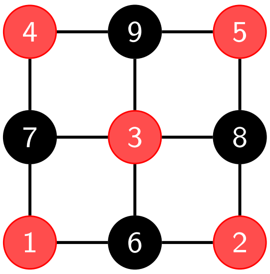



From left to right and from the bottom up

Example, pattern for $n = 6 \quad \longrightarrow$

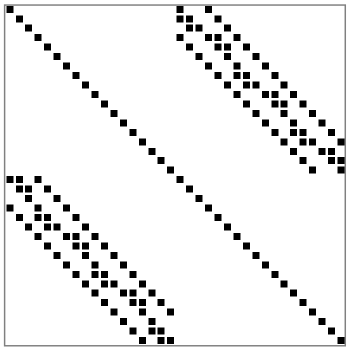


Red-Black Ordering




1. Color the nodes in a way that neighbors have different color
2. Number the red nodes first, then the black ones

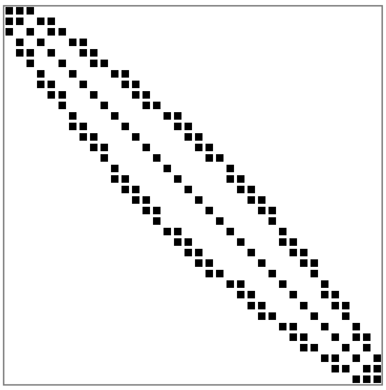
$p = [1, 3, 5, 7, 9, 2, 4, 6, 8]$



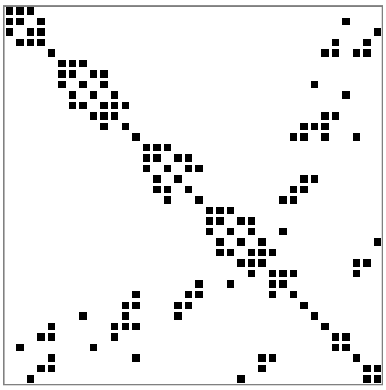


# Other Orderings

The main purpose of reordering is to improve the structural properties of the matrix



Reverse Cuthill-McKee, minimizes the bandwidth



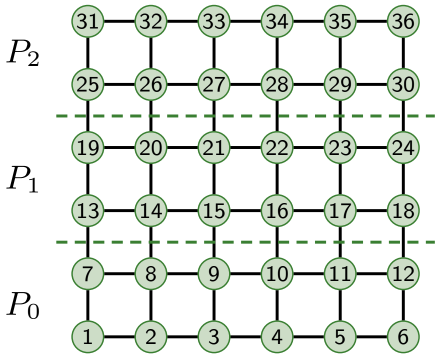
Minimum degree, reduces fill-in in the factorizations

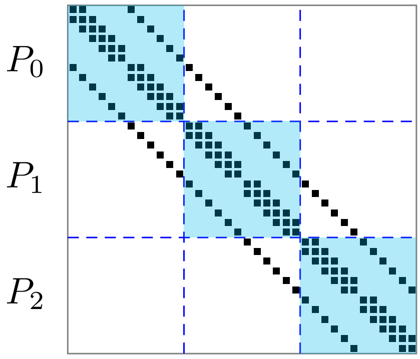
Other:

- *Nested dissection*, also reduces fill-in
- Partitioning: minimize communications in parallel

## Parallel Matrix-Vector Product

Block-row distribution with natural ordering



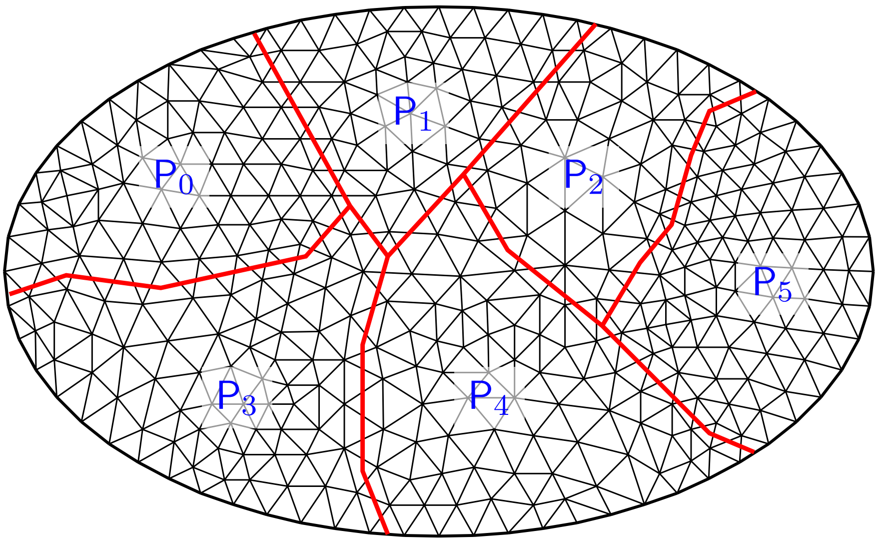

The volume of communication in parallel depends on:

- Nonzero elements outside the diagonal block
- Number of edge cuts (go from one subdomain to another)



## Mesh Partitioning

Also in unstructured meshes



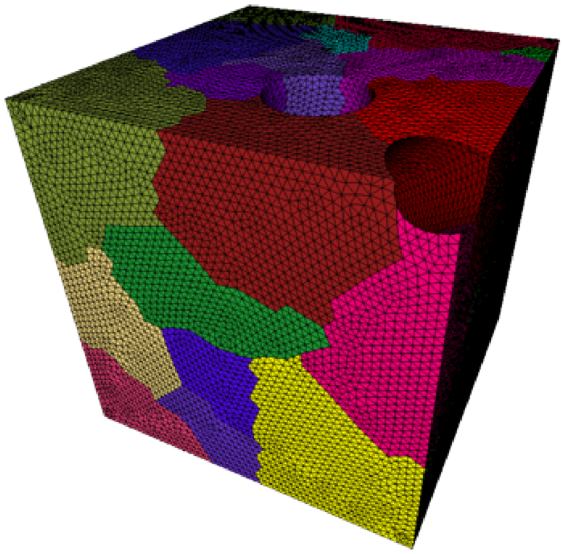

Each process sends the unknowns lying on the interface to the corresponding neighboring processes, and receives in turn from them

## Graph Partitioning

Goal: divide the mesh in regions of similar size to be assigned to each process, trying to reduce the communication cost

Types of techniques:

- Iterative exchange
- Recursive Bisection
- Multilevel



Most algorithms are based on adjacency graphs: undirected graph, with or without weights, $G = (N, E, W_N, W_E)$

- Balance the sum of weights $W_N$ in each partition
- Minimize the sum of weights $W_E$ from one partition to another



## Parallel Vectors

- For best efficiency, the position of elements to send is precomputed
- It is often necessary to have a local representation of the vector including the received values (*ghost values*)