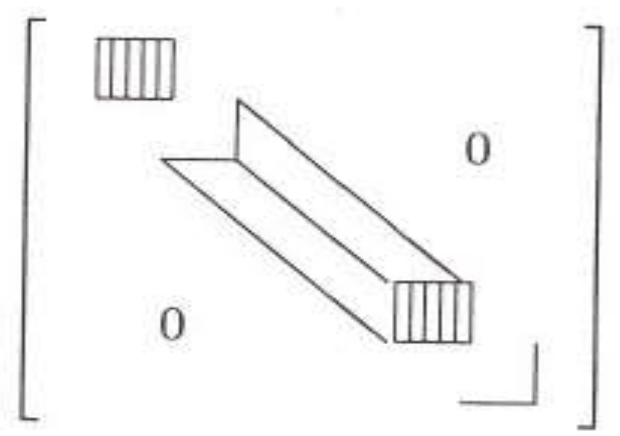
SPARSE MATRICES
DIRECT METHODS WITH GRAPH THEORY

Srilalitha Parimi
Savitri Devisetty

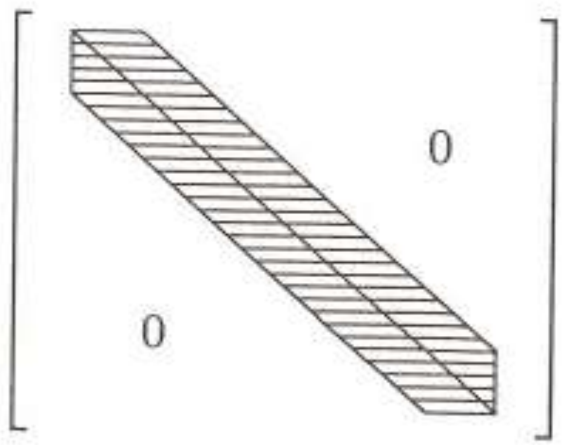
Introduction to Sparse Matrix

- A(n×n) matrix $A=[a_{ik}]$ is said to be a sparse matrix if only a small percentage of all matrix elements a_{ik}, i,k=1,2,3,4.5……,n, is nonzero

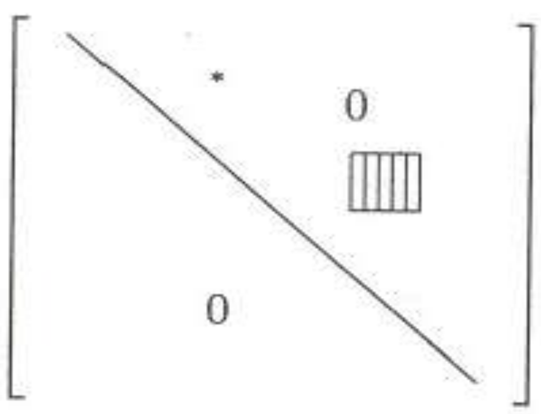
  (in practice less than 10%).



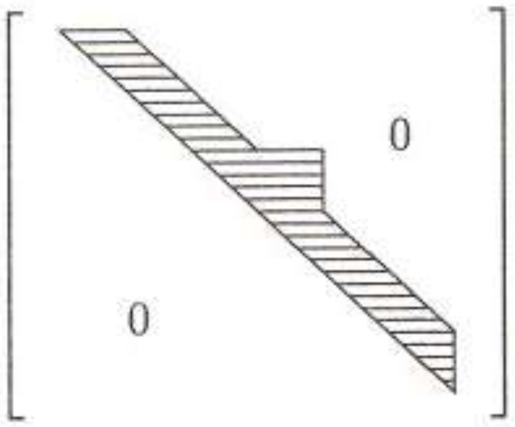
Circuit design



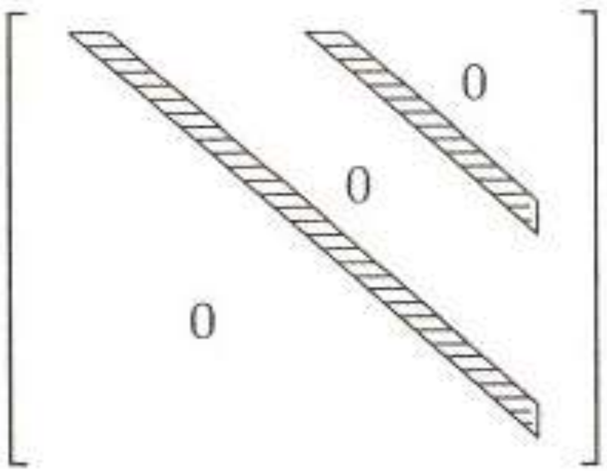
Matrix with constant bandwidth



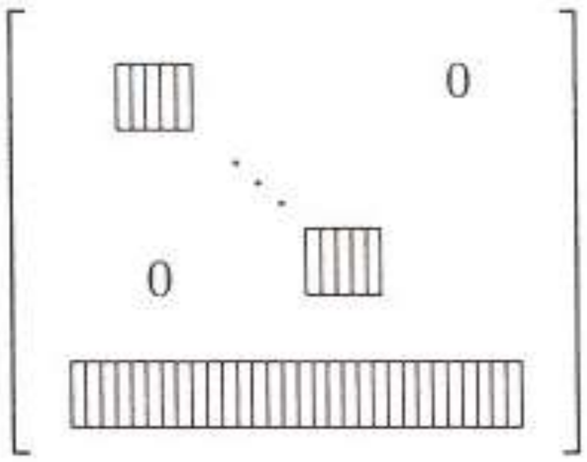
Arbitrary Sparse Matrix



Band Matrix with step



Strip Matrix



Block diagonal matrix with Margin

- Sparse Matrices
  - Why are they nice
  - How do we store them
  - How can we exploit and preserve sparsity

- The advantages of sparse matrices are size and speed.

- Consider a 5000 by 5000 matrix in full storage requires space for 25 million complex numbers even if only 50,000 are nonzero. For same matrix, sparse storage would store 50,000 complex numbers and 50,000 pairs of integer indices.

Storage of Sparse Matrix

- A sparse matrix can be stored in full-matrix storage mode or a packed storage mode.
- Packed Storage Mode
- Avoid storing zero entries
    - Memory usage reduction
    - Decomposition is faster since you do need to access them

Data Structures

- Static Data Structures
  - Compressed row Storage
  - Compressed Column Storage
  - Compressed Diagonal Storage
  - Jagged Diagonal Storage

- Dynamic Data Structures
  - Linked Lists

Compressed Row Storage

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix} .$$

val	10	-2	3	9	3	7	8	7	3… 9	13	4	2	-1		
col_ind	1	5	1	2	6	2	3	4	1… 5	6	2	5	6		

row_ptr	1	3	6	9	13	17	20

- Instead of storing n^2 elements, we need only 2nnz+n+1 storage locations.

Compressed Column Storage

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix} .$$

val	10	3	3	9	7	8	4	8	8 … 9	2	3	13	-1		
row_ind	1	2	4	2	3	5	6	3	4 … 5	6	2	5	6		

col_ptr	1	4	8	10	13	17	20

Compressed Diagonal Storage

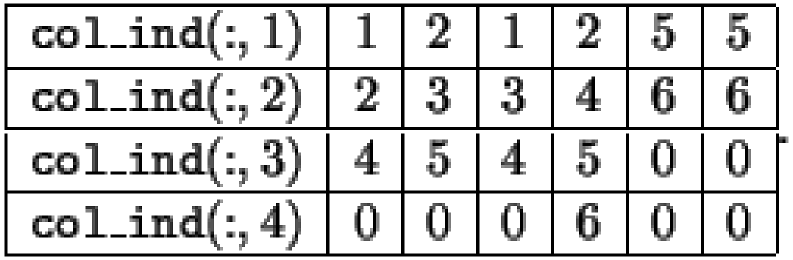
$$A = \begin{bmatrix} 10 & -3 & 0 & 0 & 0 & 0 \\ 3 & 9 & 6 & 0 & 0 & 0 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 0 & 0 & 8 & 7 & 5 & 0 \\ 0 & 0 & 0 & 9 & 9 & 13 \\ 0 & 0 & 0 & 0 & 2 & -1 \end{bmatrix} .$$

A in an array of dimension (6,-1:1) using the mapping

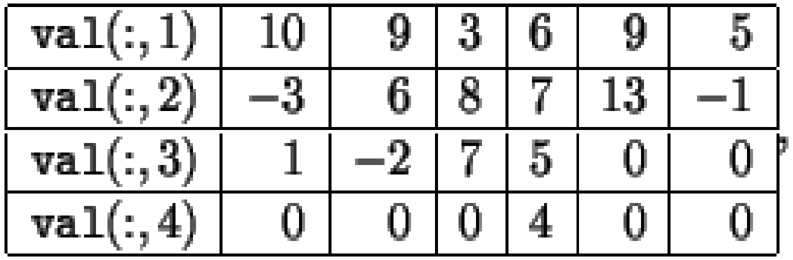
val(:,-1)	0	3	7	8	9	2				
val(:, 0)	10	9	8	7	9	-1				
val(:,+1)	-3	6	7	5	13	0				

Jagged Diagonal Storage

$$\begin{bmatrix} 10 & -3 & 0 & 1 & 0 & 0 \\ 0 & 9 & 6 & 0 & -2 & 0 \\ 3 & 0 & 8 & 7 & 0 & 0 \\ 0 & 6 & 0 & 7 & 5 & 4 \\ 0 & 0 & 0 & 0 & 9 & 13 \\ 0 & 0 & 0 & 0 & 5 & -1 \end{bmatrix} \longrightarrow \begin{bmatrix} 10 & -3 & 1 & \\ 9 & 6 & -2 & \\ 3 & 8 & 7 & \\ 6 & 7 & 5 & 4 \\ 9 & 13 & & \\ 5 & -1 & & \end{bmatrix}$$

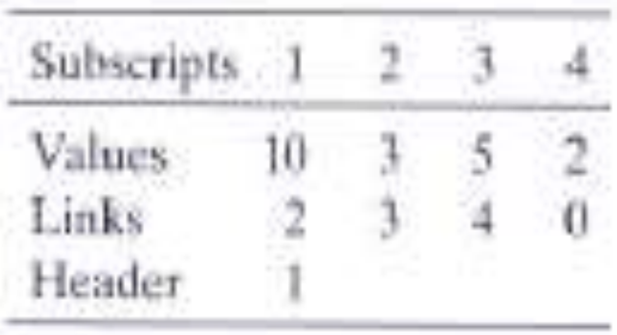


col_ind(:, 1)	1	2	1	2	5	5
col_ind(:, 2)	2	3	3	4	6	6
col_ind(:, 3)	4	5	4	5	0	0
col_ind(:, 4)	0	0	0	6	0	0



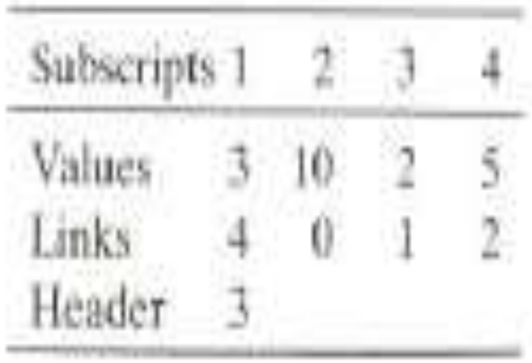
val(:, 1)	10	9	3	6	9	5
val(:, 2)	−3	6	8	7	13	−1
val(:, 3)	1	−2	7	5	0	0
val(:, 4)	0	0	0	4	0	0

- Linked List data structure is more efficient to store sparse matrices.



Subscripts	1	2	3	4
Values	10	3	5	2
Links	2	3	4	0
Header	1			

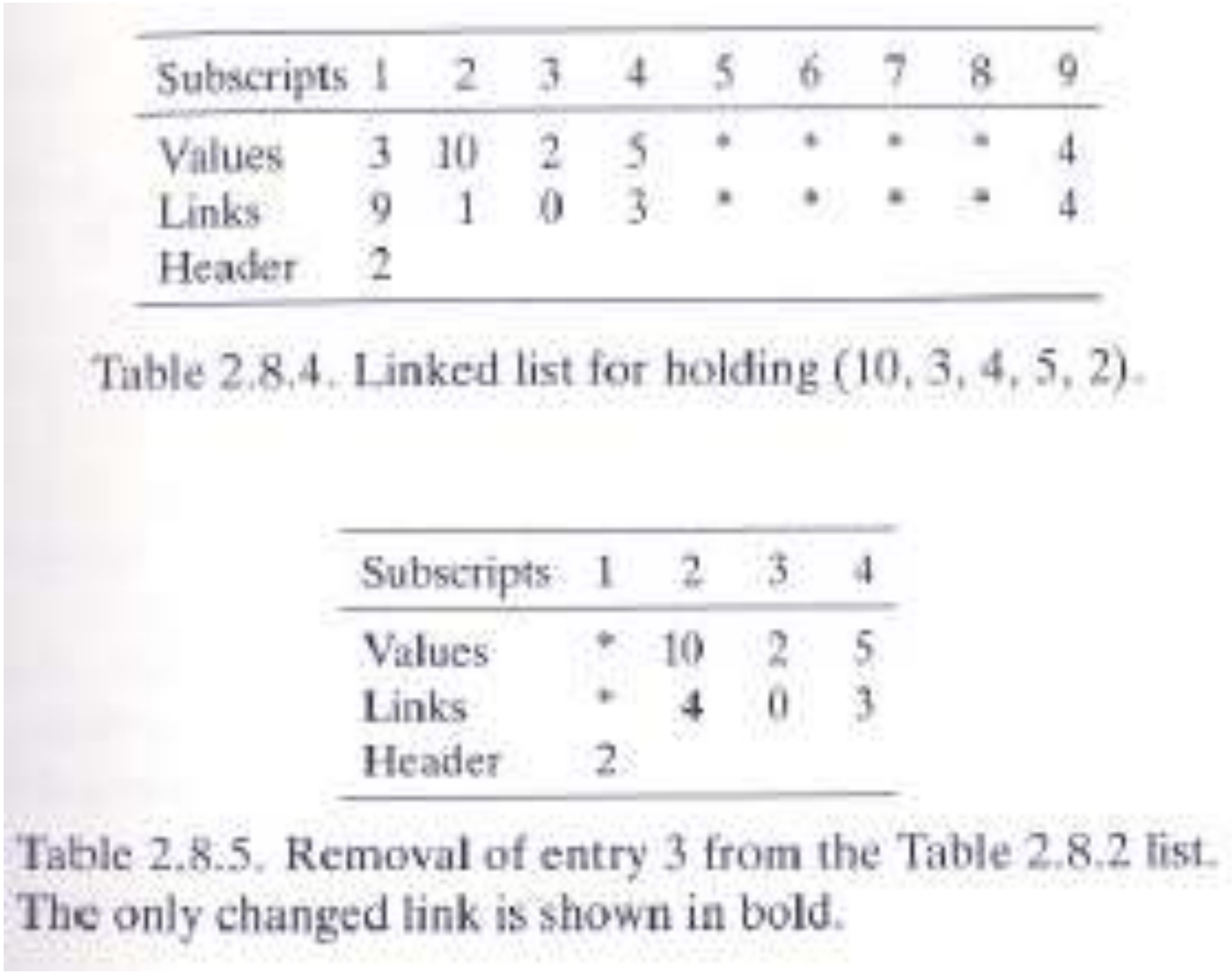
Table 2.8.1. Linked list for holding (10, 3, 5, 2).



Subscripts	1	2	3	4
Values	3	10	2	5
Links	4	0	1	2
Header	3			

Table 2.8.3. Linked list for holding (10, 3, 5, 2) in increasing order of values.

- Inserting and deleting values from linked list.



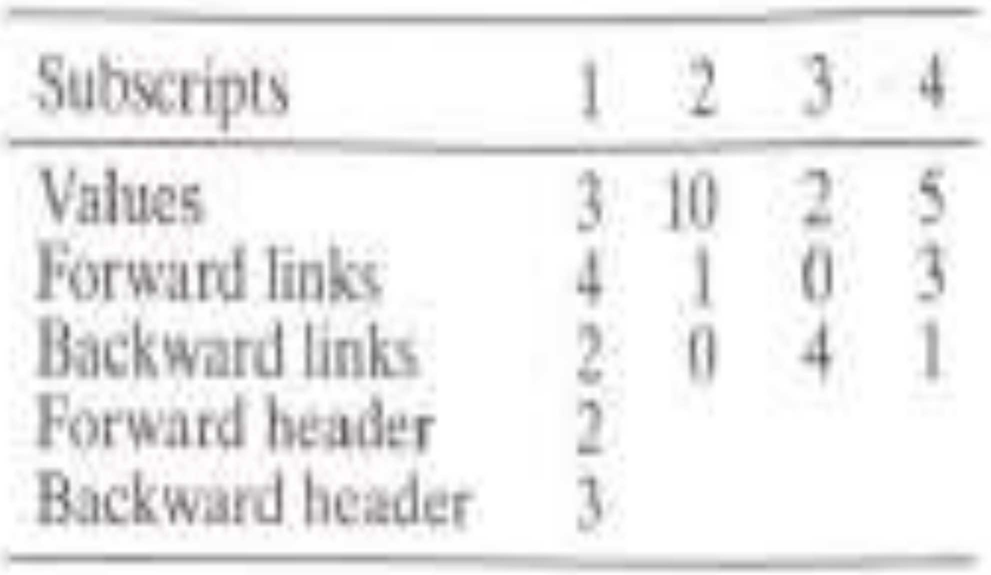
Subscripts	1	2	3	4	5	6	7	8	9
Values	3	10	2	5	*	*	*	*	4
Links	9	1	0	3	*	*	*	*	4
Header	2								

Table 2.8.4. Linked list for holding (10, 3, 4, 5, 2).

Subscripts	1	2	3	4
Values	*	10	2	5
Links	*	**4**	0	3
Header	2			

Table 2.8.5. Removal of entry 3 from the Table 2.8.2 list. The only changed link is shown in bold.

- Using Doubly-linked list to store the sparse matrix.



Subscripts	1	2	3	4
Values	3	10	2	5
Forward links	4	1	0	3
Backward links	2	0	4	1
Forward header	2			
Backward header	3			

Table 2.8.6. Doubly-linked list holding (10, 3, 5, 2).

Sparse Matrix --- Graph Approach

- Graph Theory is a visualize tool what is happening in sparse matrix computation.
- A graph G(V,E) is a set of vertices V and set of edges E between nodes.

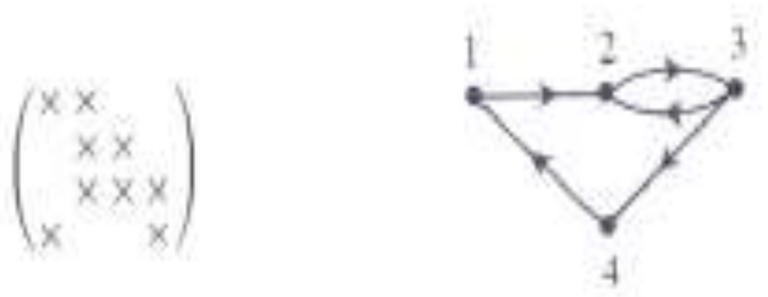


Figure 1.2.1. An unsymmetric matrix and its digraph.

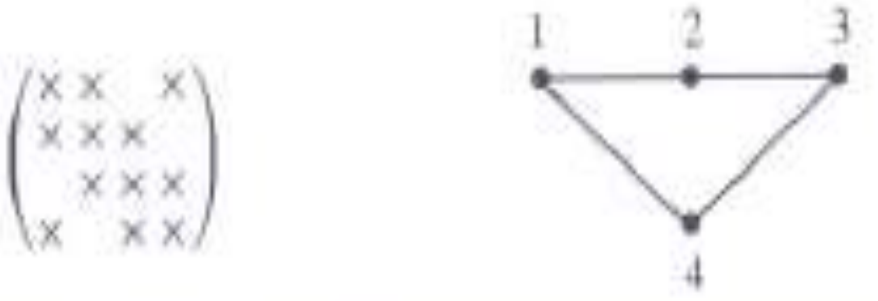


Figure 1.2.2. A symmetric matrix and its graph.

DIRECT METHODS

Advantages

- High accuracy
- Easy to package
- Method of choice in many applications
- Not dramatically affected by conditioning
- Reasonably independent of structure

Gaussian Elimination

$$PAQ \rightarrow LU$$

Permutations P and Q chosen to preserve sparsity and maintain stability

L : Lower triangular (sparse)
U : Upper triangular (sparse)

SOLVE:

$$Ax = b$$

by

$$Ly = Pb$$

then

$$UQx = y$$

Reduction to block triangular form

- Allows the corresponding set of linear equations to be solved as a sequence of sub problems.
- It saves both computational work and storage.

$$\mathbf{PAQ} = \begin{pmatrix} \mathbf{B}_{11} & & & & \\ \mathbf{B}_{21} & \mathbf{B}_{22} & & & \\ \mathbf{B}_{31} & \mathbf{B}_{32} & \mathbf{B}_{33} & & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \mathbf{B}_{N1} & \mathbf{B}_{N2} & \mathbf{B}_{N3} & \cdots & \mathbf{B}_{NN} \end{pmatrix}.$$

- Economical algorithms requires O(n) +O(t) operations for a matrix of order n with t entries.

Depth-first search algorithm for traversal extension

- First entry in column *k* is in row *j.* If column *j* has an entry in row *k* or beyond, say in row *p,* this means that an interchange between rows *p* and *j* will preserve the (j, *j)* entry and move the entry in column *k* to row *p>= k.*

$$
\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & \times & & & & & \\
2 & & \times & & & & \times \\
3 & & & \times & & & \\
4 & & \times & & \times & & \\
5 & & & & & \times & \\
6 & & & & \times & & \\
\end{array}
\quad \text{becomes} \quad
\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & \times & & & & & \\
4 & & \times & & \times & & \\
3 & & & \times & & & \\
6 & & & & \times & & \\
5 & & & & & \times & \\
2 & & \times & & & & \times \\
\end{array}
$$

Figure 6.3.1. Two row interchanges needed in the transversal algorithm.

- If the matrix has t entries the number of elementary operations is at most proportional to nt.

Symmetric permutations to block triangular form

- Causes no change in the associated digraph except for the relabelling of its nodes.
- Sargent and Westerberg algorithm
- Tarjan’s algorithm

Sargent and Westerberg algorithm

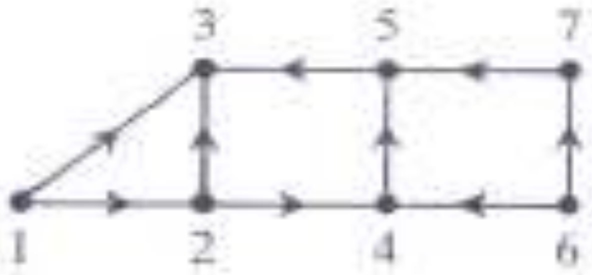


Figure 6.7.1. A digraph corresponding to a triangular matrix.

Path					**5**						
			3	4	4	**4**					
		2	2	2	2	2	**2**			**7**	
	1	1	1	1	1	1	1	**1**	6	6	**6**
Step	1	2	3	4	5	6	7	8	9	10	11

Figure 6.7.2. The sequence of paths used for the Figure 6.7.1 case, where nodes selected for ordering are shown in bold.

- After completing the process graph can be relabeled like this:

  3 → 1,5 → 2, 4 → 3, 2 → 4, 1 → 5, 7 → 6, 6 → 7

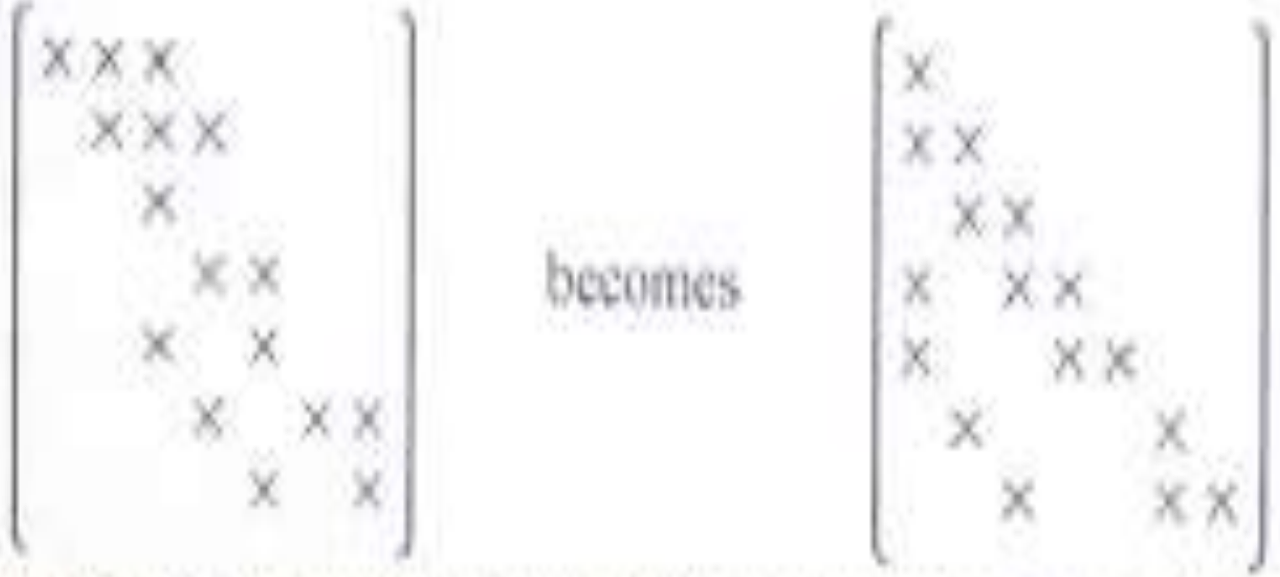


Figure 6.7.3. The matrices before and after renumbering.

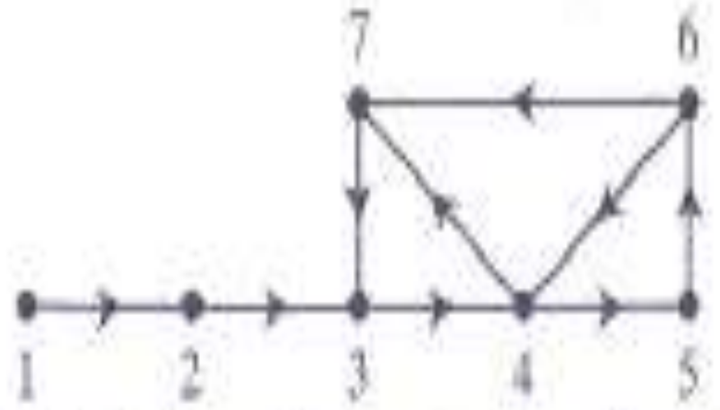


Figure 6.7.4. A digraph illustrating the algorithm of Sargent and Westerberg.

• Each composite node with the lowest label of its constituent nodes can take O(n^2) relabellings.

•Disadvantage of this algorithm is large overheads associated with the relabelling in case of difficult graphs.

Tarjan’s algorithm

- Same idea of previous algorithm but stack is used to record the current paths and all the closed paths.

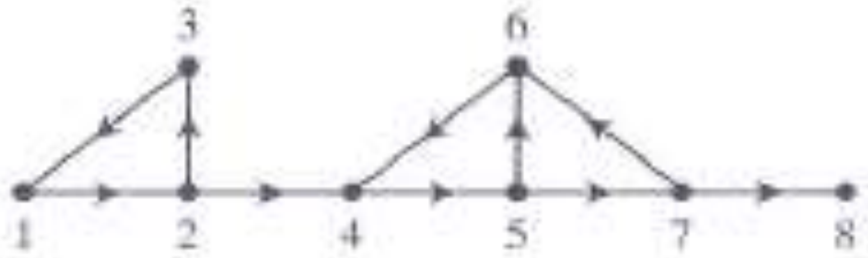


Figure 6.8.2. An example with two nontrivial strong components.

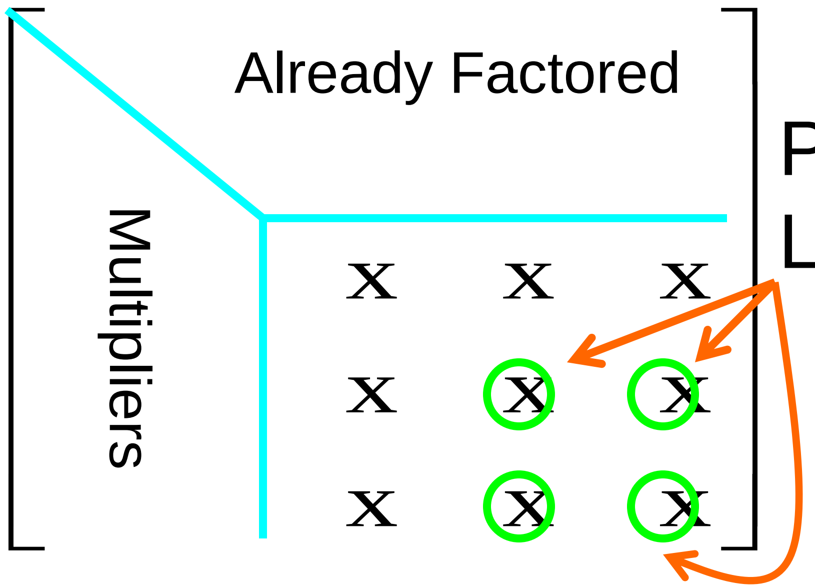
										8					
									7	7	7_6	7	7		
							6_4	6	6	6	6	6	6		
						5	5	5_4	5_4	5_4	5_4	5_4	5		
Stack					4	4	4	4	4	4	4	4	4		
			3_1	3	3	3	3	3	3	3	3	3	3	3	3
		2	2	2_1	2_1	2_1	2_1	2_1	2_1	2_1	2_1	2_1	2_1	2_1	2
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Step	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 6.8.3. The stack corresponding to Figure 6.8.2.

- It takes O(n) costs associated with initialization and accessing the rows.
- The overall cost is O(n)+O(t) for a matrix of order n with t entries.
- Improved computational cost : factor in $O(N^{1.5})$ operations (dense is $O(N^3)$)

Sparse matrices- fill in and reordering

Where can fill-in occur ?



Possible Fill-in Locations

Fill-in Estimate = (Non zeros in unfactored part of Row -1)* (Non zeros in unfactored part of Col -1)

Strategies for reordering to reduce fill-in

- Local strategies
  - Markowitz algorithm
  - Minimum degree or Tinney and Walker
- Desirable forms
  - Cuthill McKee reordering
  - Reverse Cuthill McKee reordering
  - Refined quotient trees

LOCAL STRATEGIES

Markowitz algorithm

Markowitz product $(r_i - 1) * (c_j - 1)$

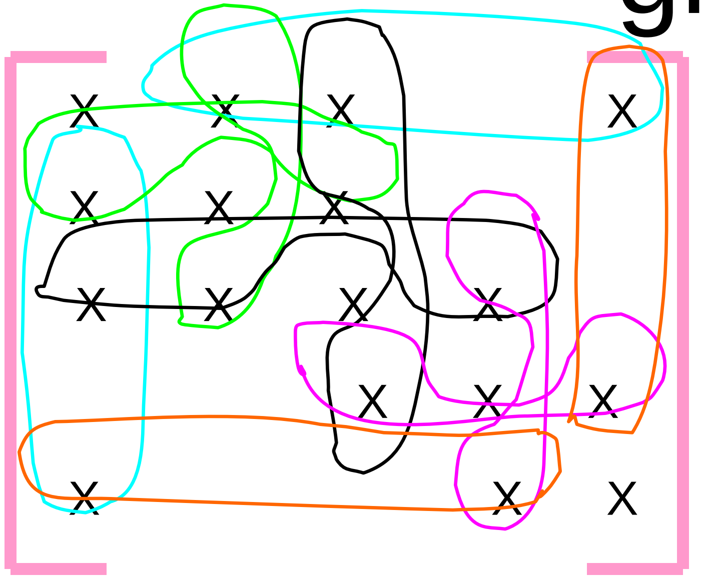
For i to n

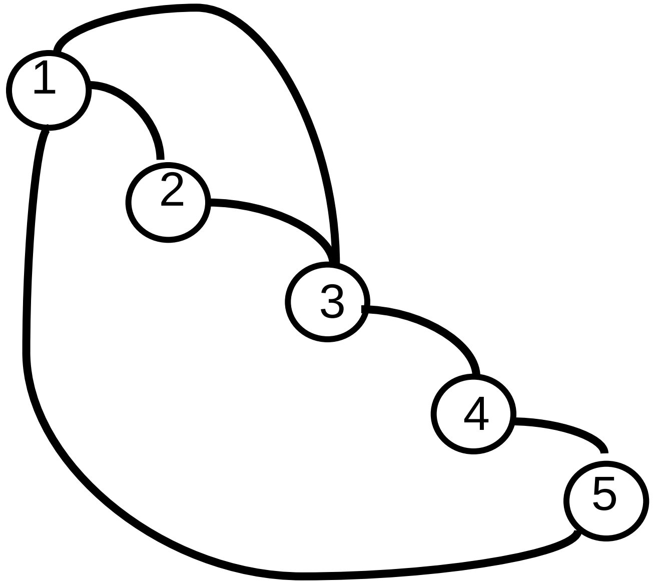
- Find diagonal j>=i with min Markowitz Product
- Swap rows j!=i and columns j!=I
- Factor the new row i and determine fill-ins

Markowitz has 5% more fill ins than the others but is faster

Greedy Algorithm (but close to optimal) !

Markowitz reordering using graphs

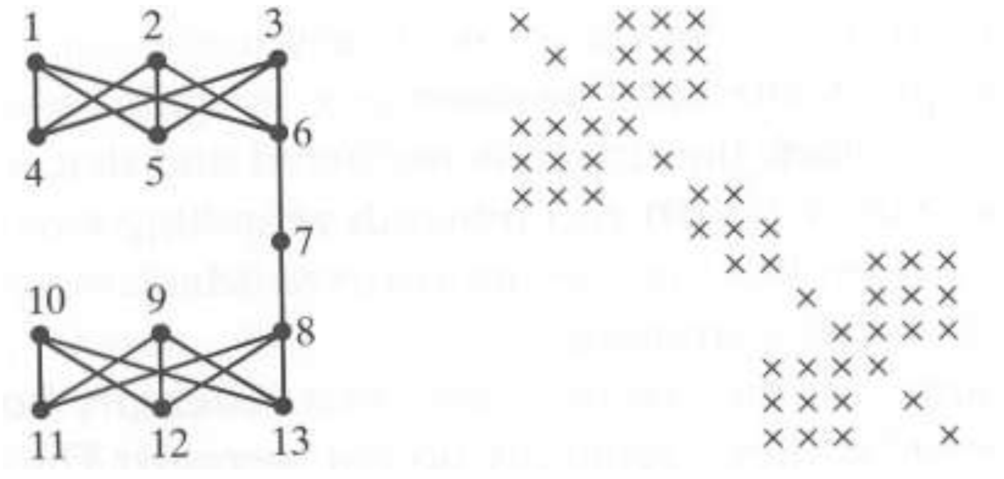




Markowitz Products = (Node Degree)^2

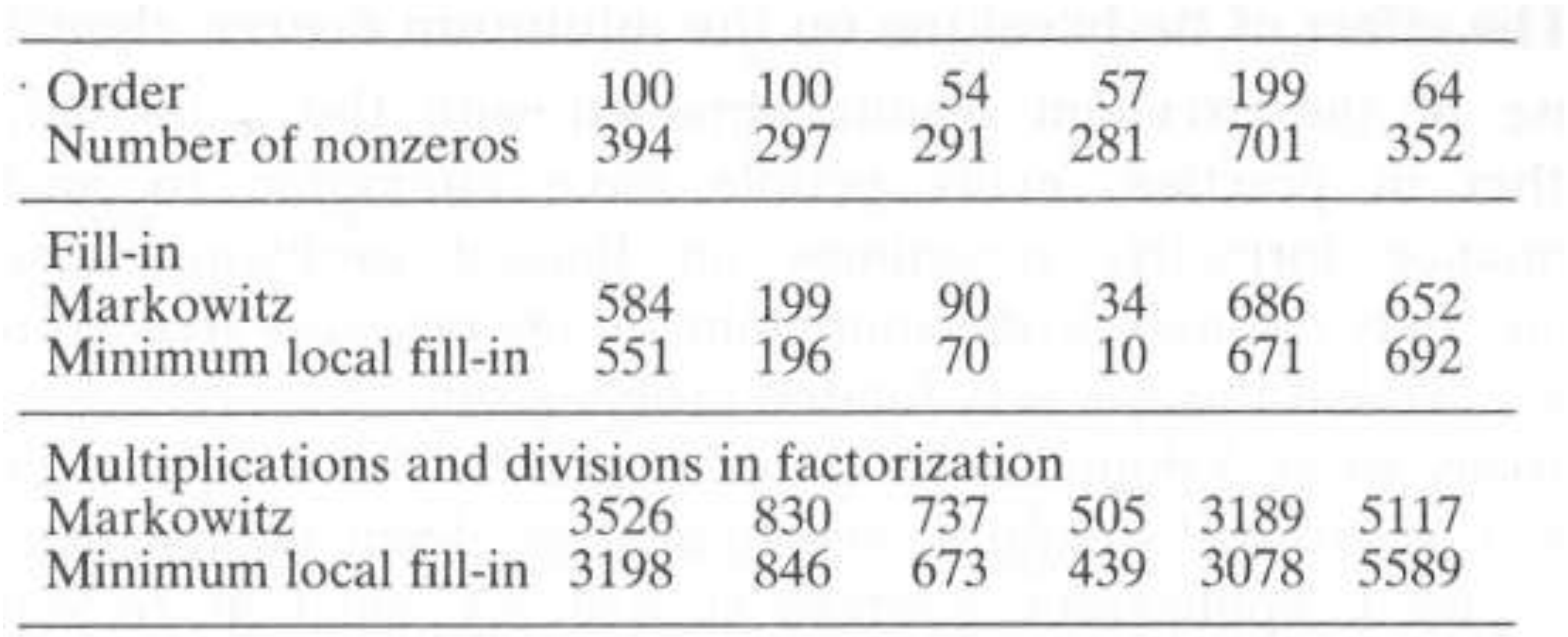
Local minimum fill in

- In the kth stage of Gaussian elimination, select a pivot which is non zero that introduces least fill in at this stage
- Doesn't always produce minimum fill in globally



Minimum fill in not optimal

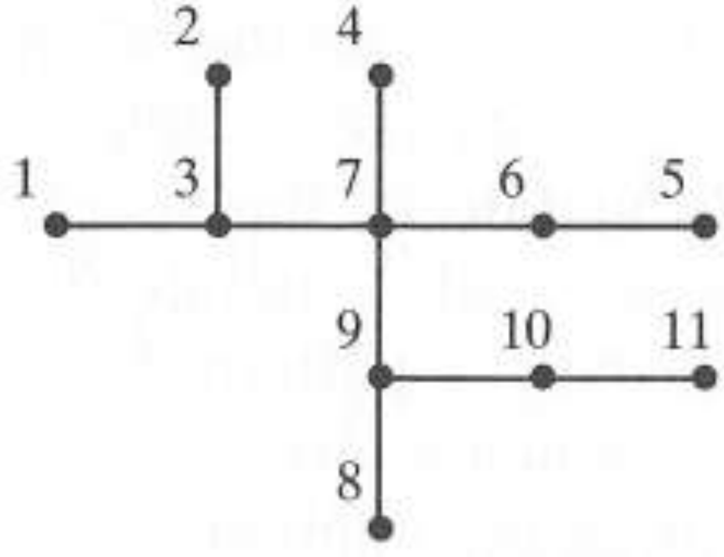
Comparison between Markowitz and minimum fill in



Order	100	100	54	57	199	64
Number of nonzeros	394	297	291	281	701	352
Fill-in						
Markowitz	584	199	90	34	686	652
Minimum local fill-in	551	196	70	10	671	692
Multiplications and divisions in factorization						
Markowitz	3526	830	737	505	3189	5117
Minimum local fill-in	3198	846	673	439	3078	5589

Minimal degree Tinney/Walker reordering

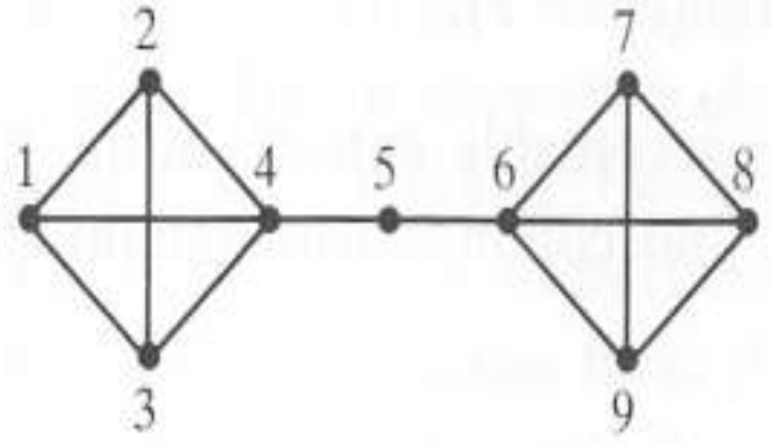
- Special case of Markowitz for symmetric matrices
- In the graph of a symmetric matrix choose the node for elimination which has least edges connected to it
- Least fill in for tree graphs



Tree graph

Minimum degree Tinney/Walker reordering

- Efficient implementation because
  - Diagonal is stable as numerical pivot
- Doesn’t minimize fill in the case of closed graphs

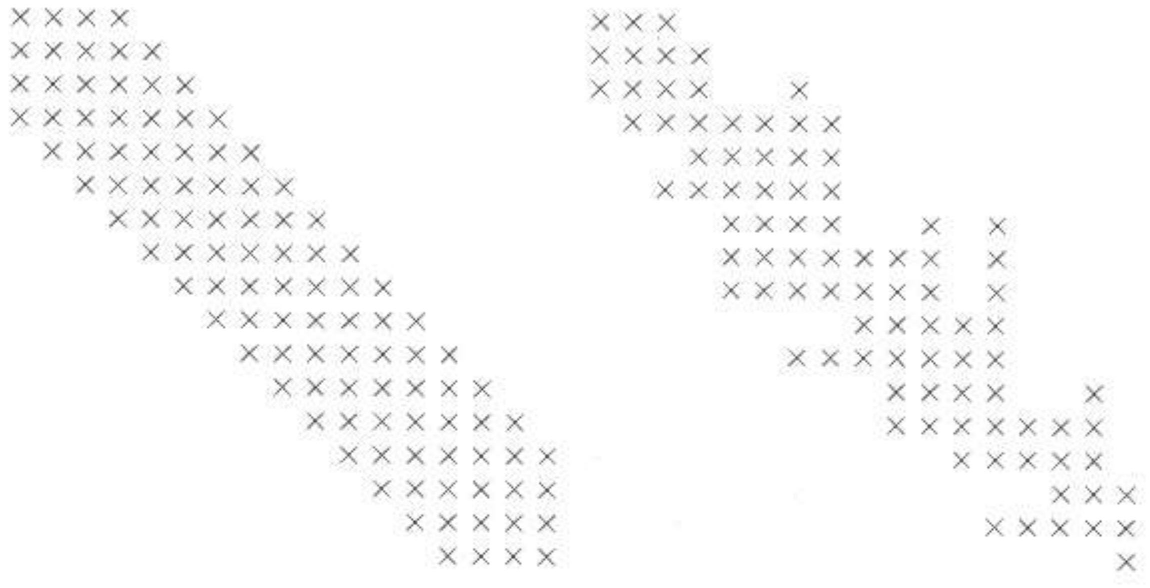


$$\begin{array}{ccccccccc}
\times & \times & \times & \times & & & & & \\
\times & \times & \times & \times & & & & & \\
\times & \times & \times & \times & & & & & \\
\times & \times & \times & \times & \times & & & & \\
 & & & \times & \times & \times & & & \\
 & & & & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times
\end{array}$$

Minimum degree not optimal

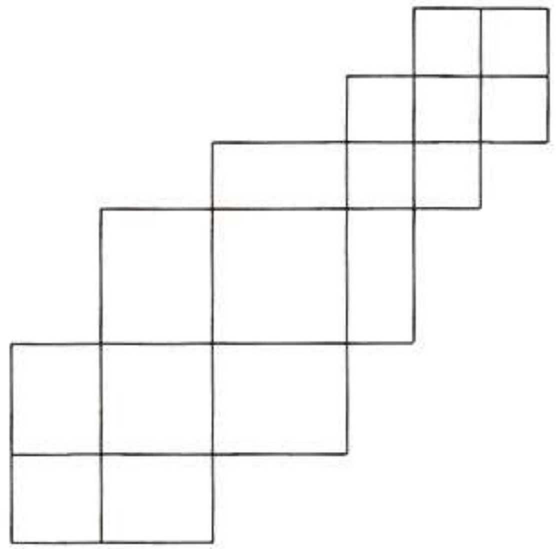
DESIRABLE FORMS

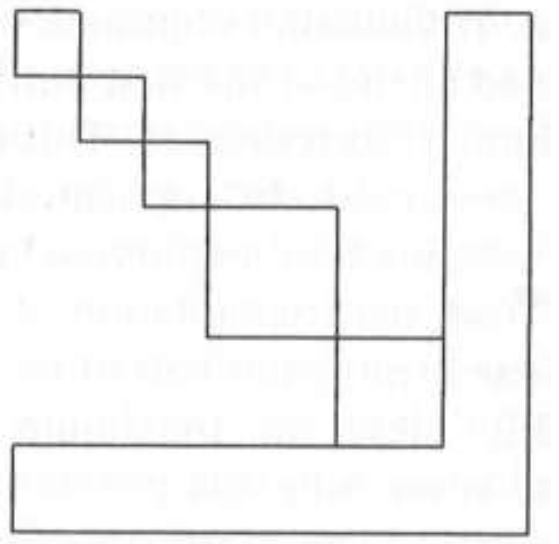
Most desired forms

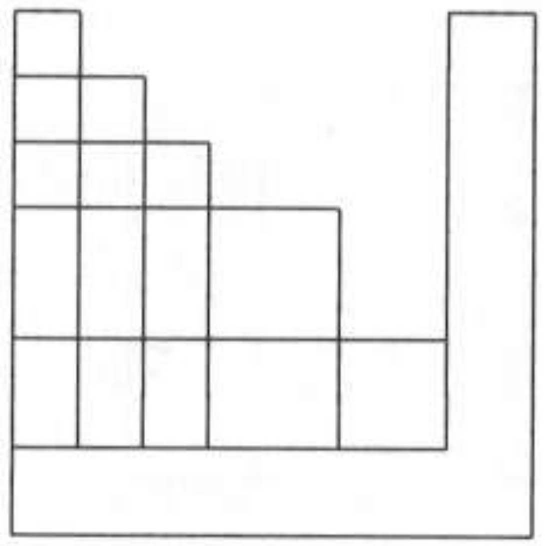


Band and variable band matrices

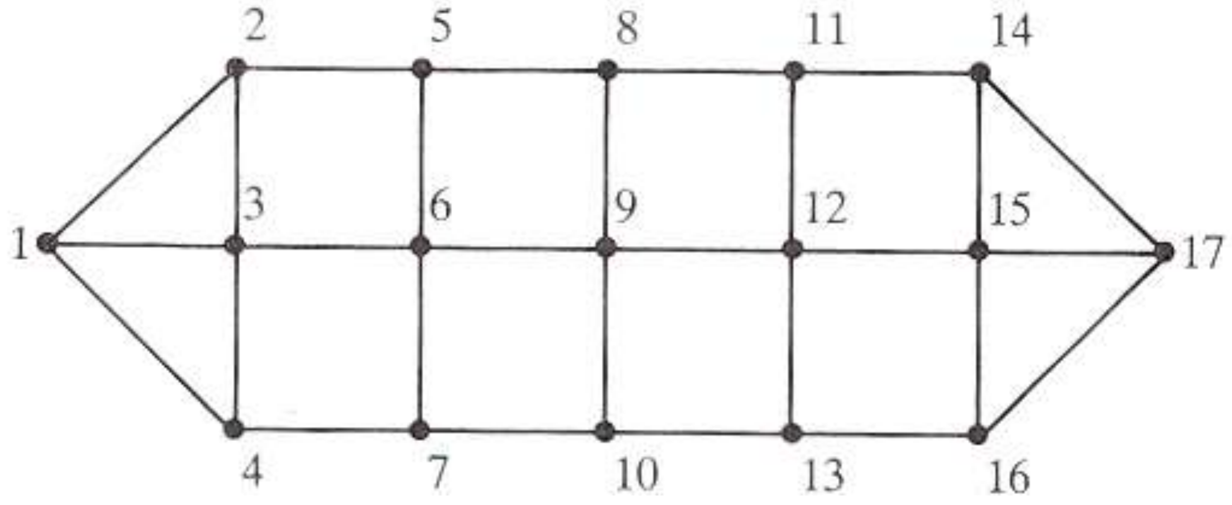
Desired forms


Block tridiagonal matrix


Doubly bordered block tridiagonal matrix

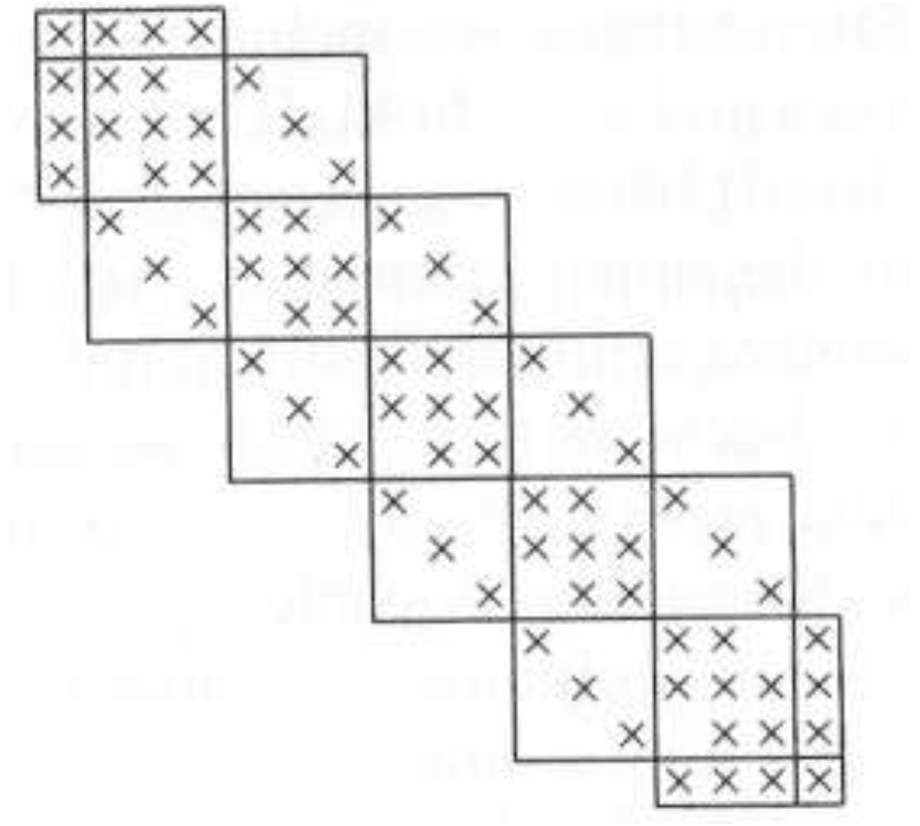

Bordered block tridiagonal matrix

Cuthill and McKee Ordering



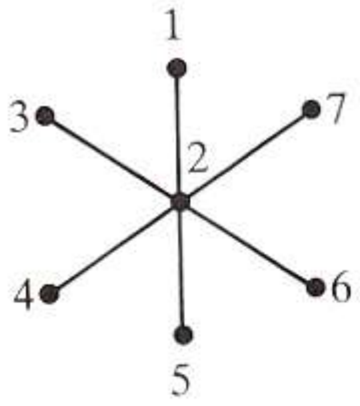
- Divide the nodes into level sets S_i with S_1 consisting of a single node
- S_2 consists of all neighbors of this node in S_1
- S_3 consists of all neighbors of nodes in S_2 that are not in S_1 or S_2

Cuthill and McKee Reordering



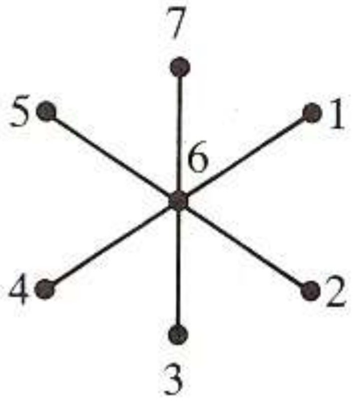
Matrix form after Cuthill and McKee reordering

Reverse Cuthill McKee reordering



$$\begin{array}{ccccccc}
\times & \times & & & & & \\
\times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & 0 & 0 & 0 & 0 \\
 & \times & 0 & \times & 0 & 0 & 0 \\
 & \times & 0 & 0 & \times & 0 & 0 \\
 & \times & 0 & 0 & 0 & \times & 0 \\
 & \times & 0 & 0 & 0 & 0 & \times
\end{array}$$

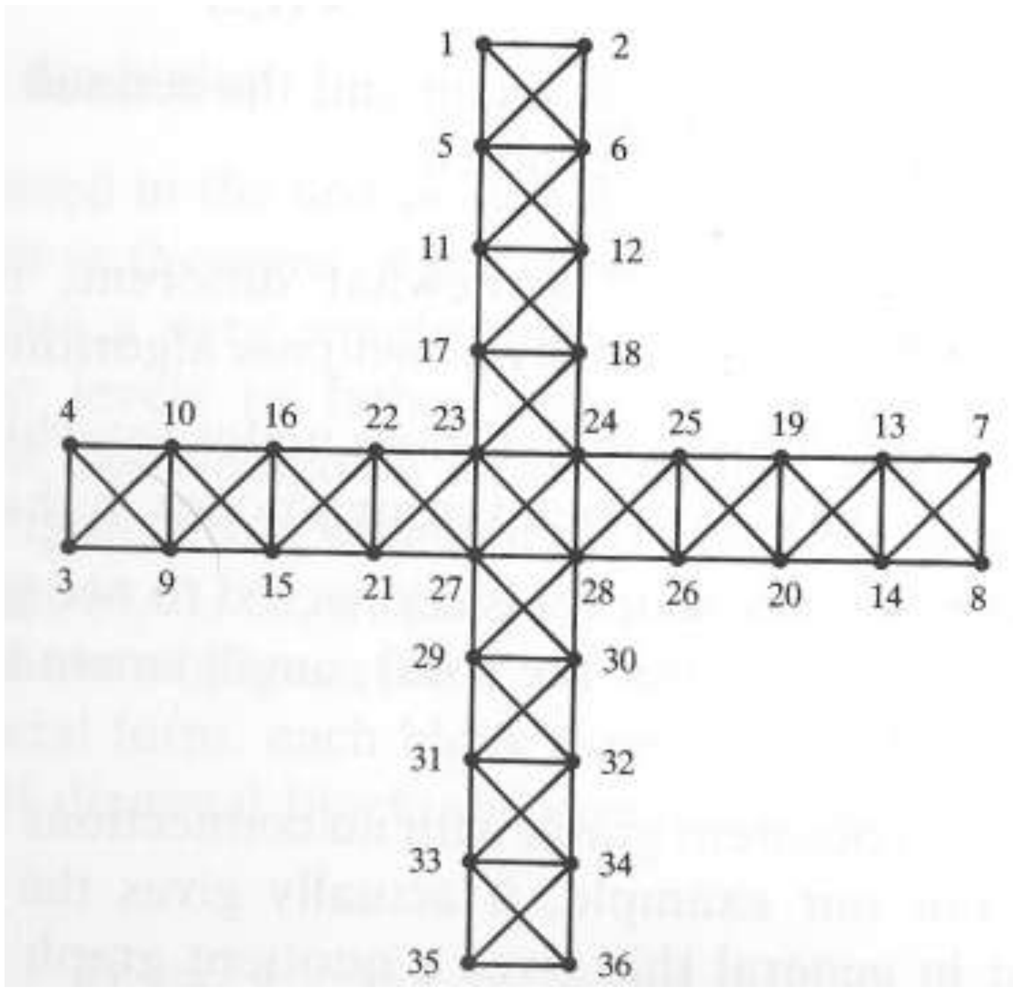
Cuthill McKee ordering



$$\begin{array}{ccccccc}
\times & & & & & \times & \\
 & \times & & & & \times & \\
 & & \times & & & \times & \\
 & & & \times & & \times & \\
 & & & & \times & \times & \\
\times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times
\end{array}$$

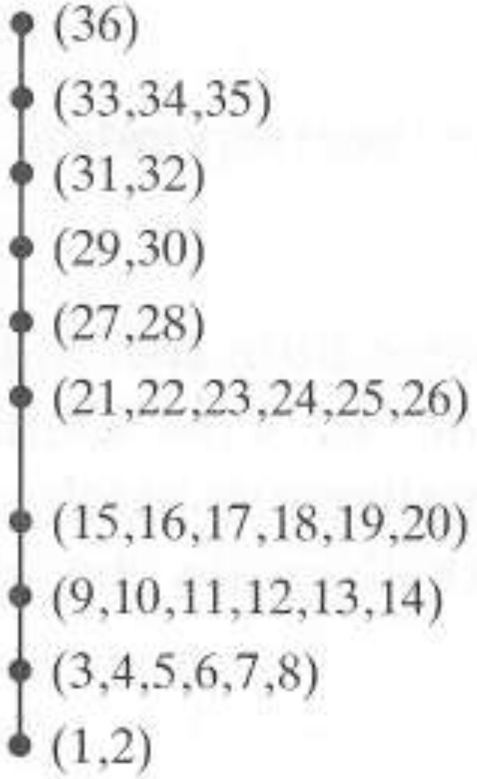
Reverse Cuthill McKee ordering

Refined quotient trees

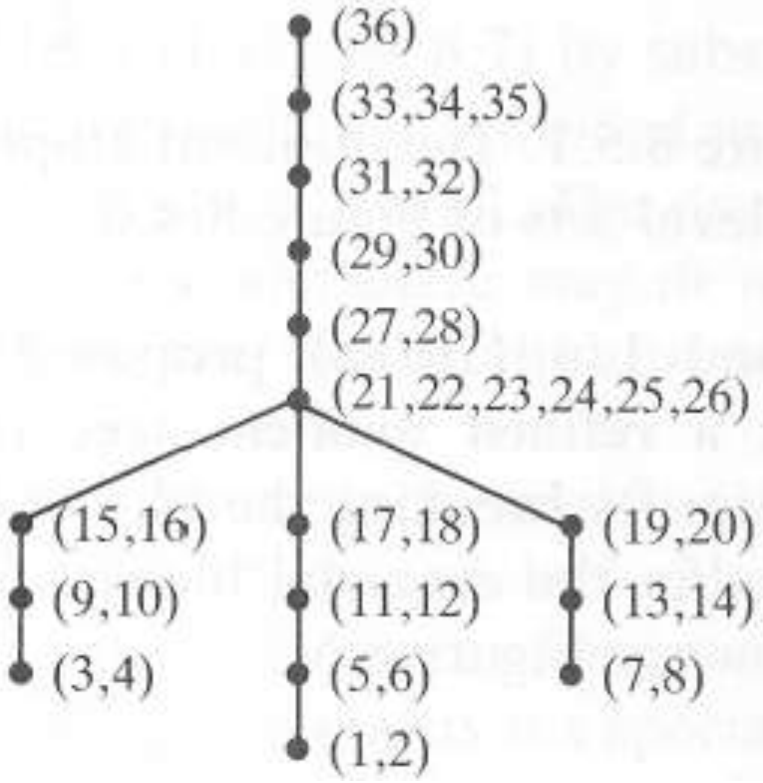


Plus shaped problem, ordered by Reverse Cuthill-McKee

Refined quotient tree



Level set chain

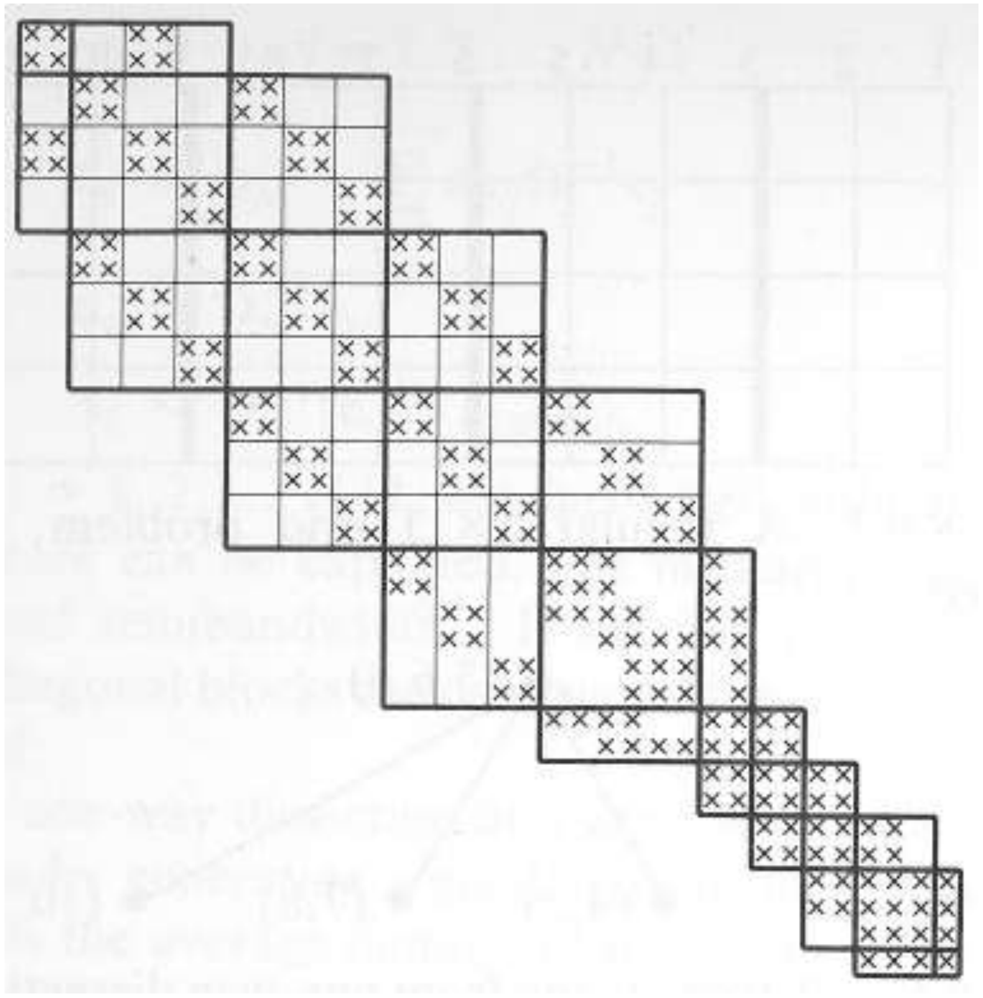


Refined quotient tree

Algorithm to build refined quotient tree

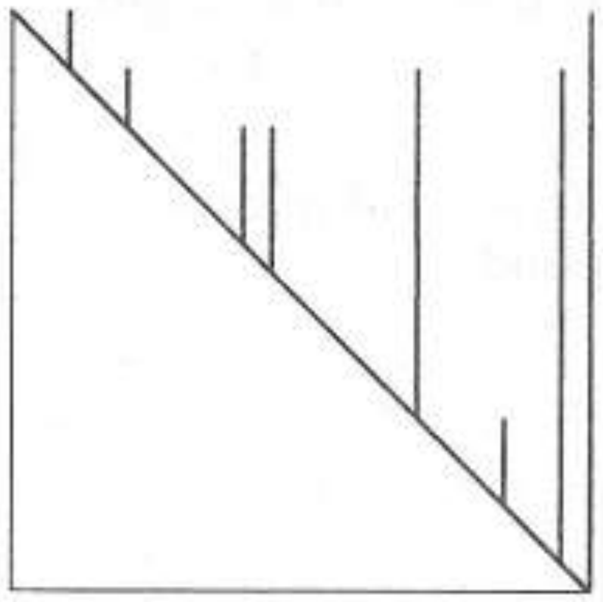
- In each level set, group together those nodes for which there is a path from any one to any other one through nodes in the level sets
- Whenever a group at one level is connected to two groups at the next higher level, amalgamate the two groups at the higher level

Refined quotient tree

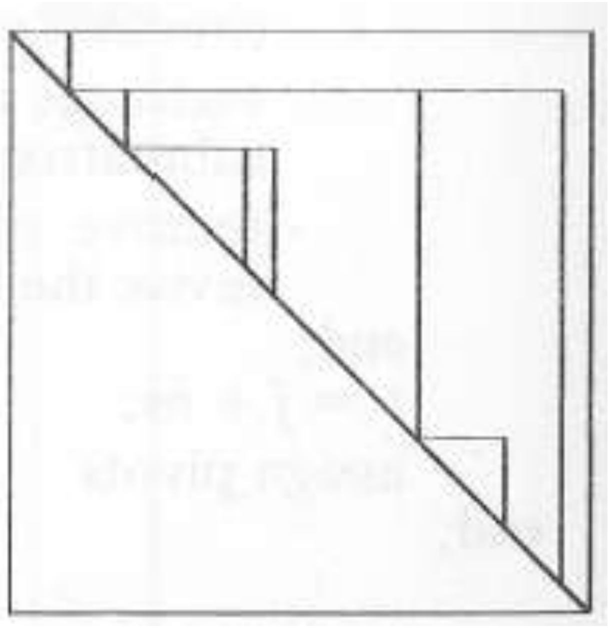


Block matrix corresponding to the tree

Hellerman – Rarick reordering



Spiked matrix

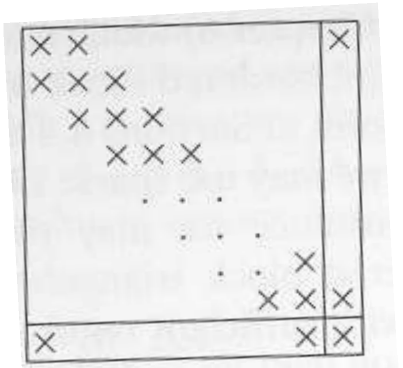


Nested bordered block triangular form of the spiked matrix

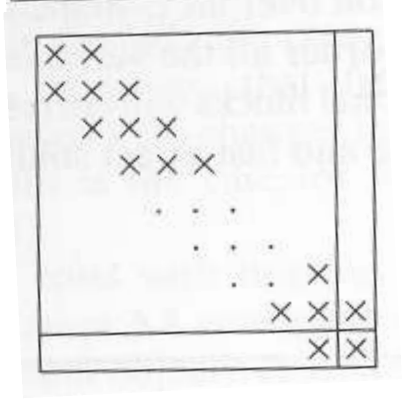
TECHNIQUES FOR SOLVING HUGE MATRICES

Partitioning

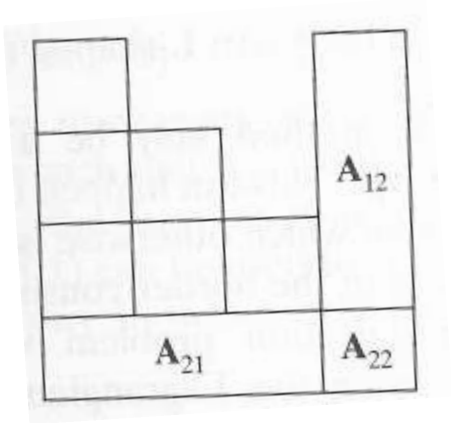
Using $\begin{pmatrix} \mathbf{L}_{11} & \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ & \mathbf{U}_{22} \end{pmatrix}$ Will result in L21 and U12 being dense



Using $\begin{pmatrix} \mathbf{A}_{11} & \\ \mathbf{A}_{21} & \bar{\mathbf{A}}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \bar{\mathbf{U}}_{12} \\ & \mathbf{I} \end{pmatrix}$ Will result in



Using $\begin{pmatrix} \mathbf{I} & \\ \ddot{\mathbf{L}}_{21} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ & \ddot{\mathbf{A}}_{22} \end{pmatrix}$ Will result in



Usually used ones are where A11 is block triangular

Perturbation to solve linear equations

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 2 \end{pmatrix} .$$

Original matrix

$$\Delta\mathbf{A} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} (1) \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Perturbation

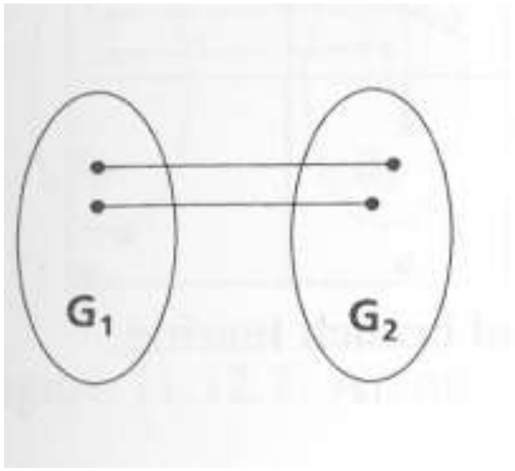
$$\mathbf{A} - \Delta\mathbf{A} = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix}$$

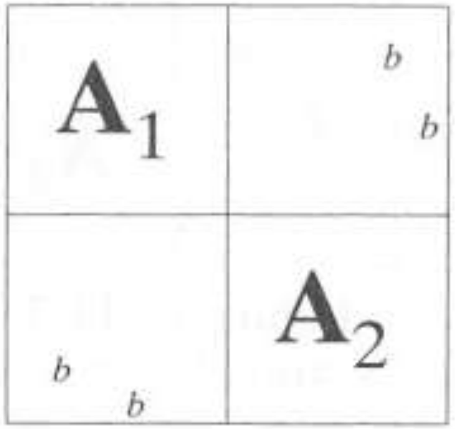
Modified matrix

$$\begin{pmatrix} 1 & & & & & & 1 \\ & 1 & & & & & 1 \\ & & 1 & & & & 1 \\ & & & 1 & & & 1 \\ & & & & 1 & & 1 \\ & & & & & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} \\ \mathbf{x} \\ \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \\ \mathbf{b} \\ \\ \mathbf{0} \end{pmatrix}$$

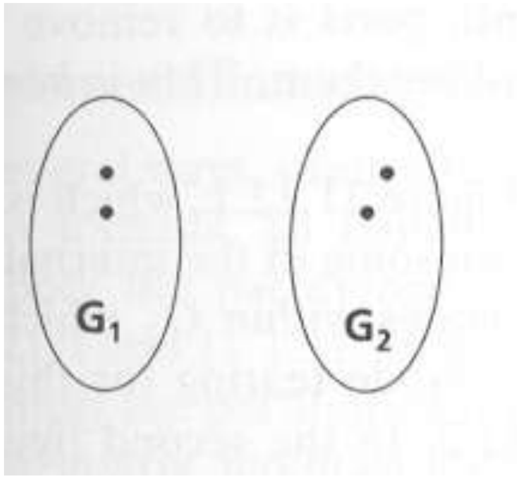
Final equation to solve

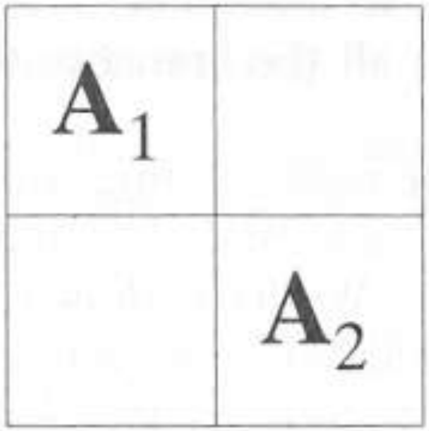
Branch tearing





Connected graphs



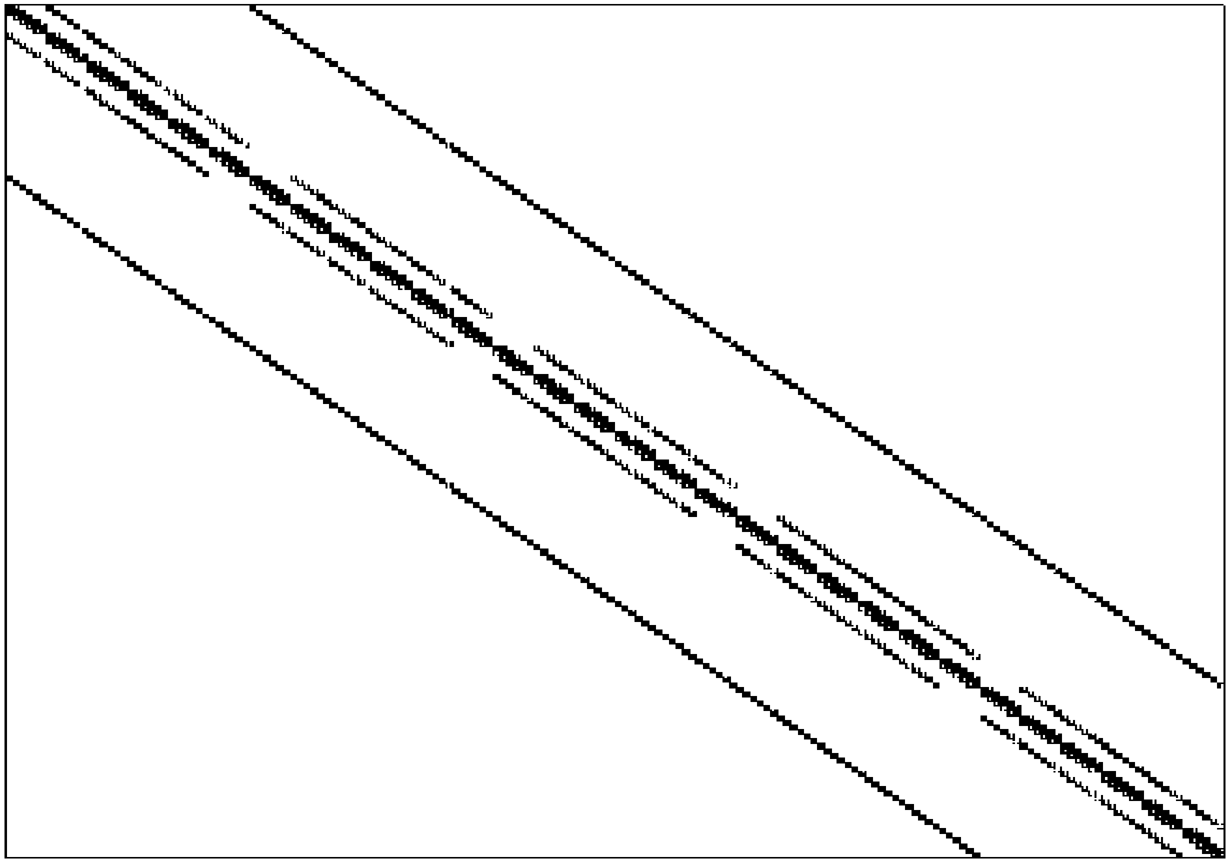


Torn apart problem

Applications

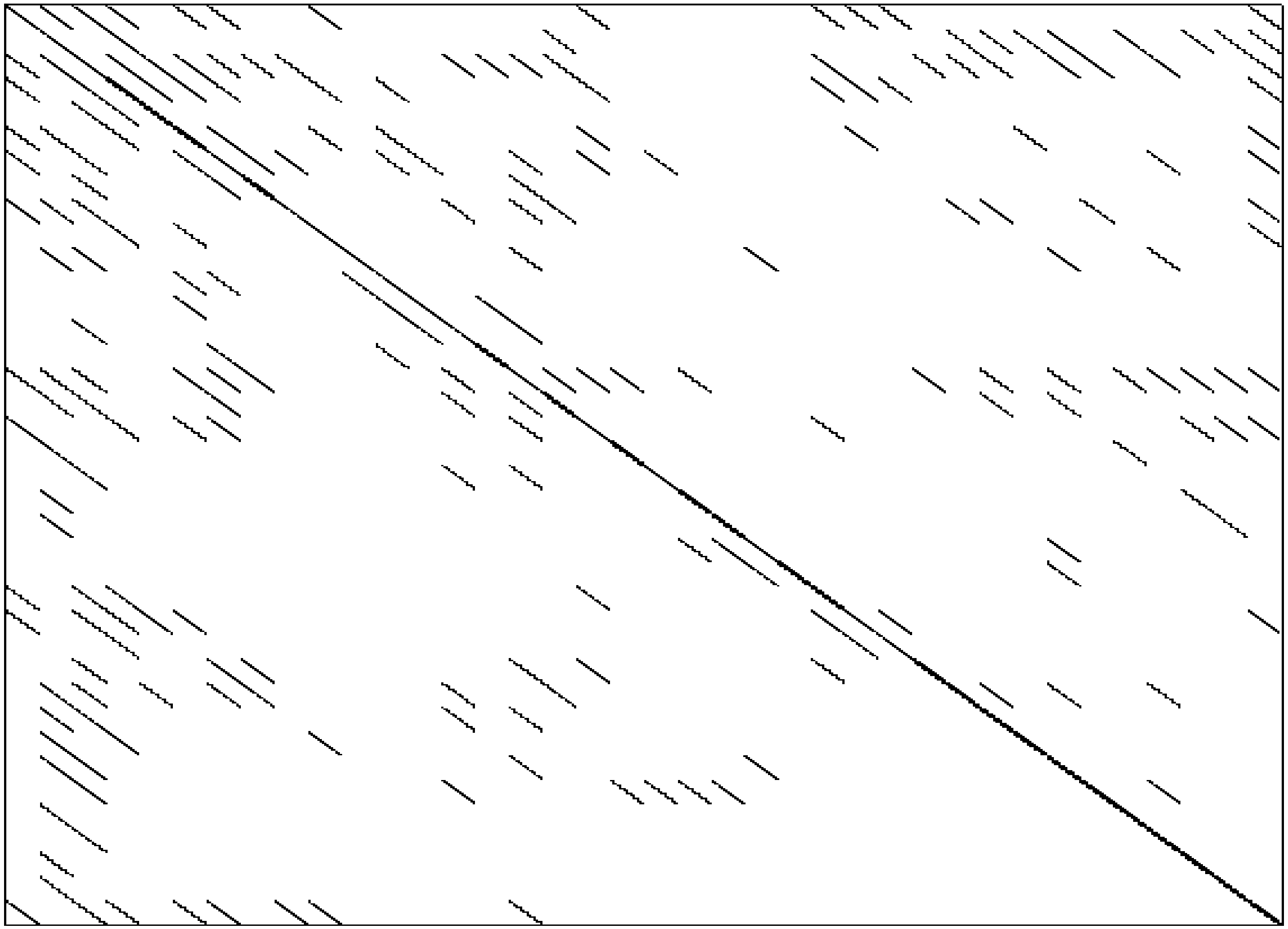
- Differential equations
- Linear Programming … Simplex
- Optimization/Nonlinear Equations
- Eigen system Solution
- Two Point Boundary Value Problems
- Least Squares Calculations

Thermal Simulation

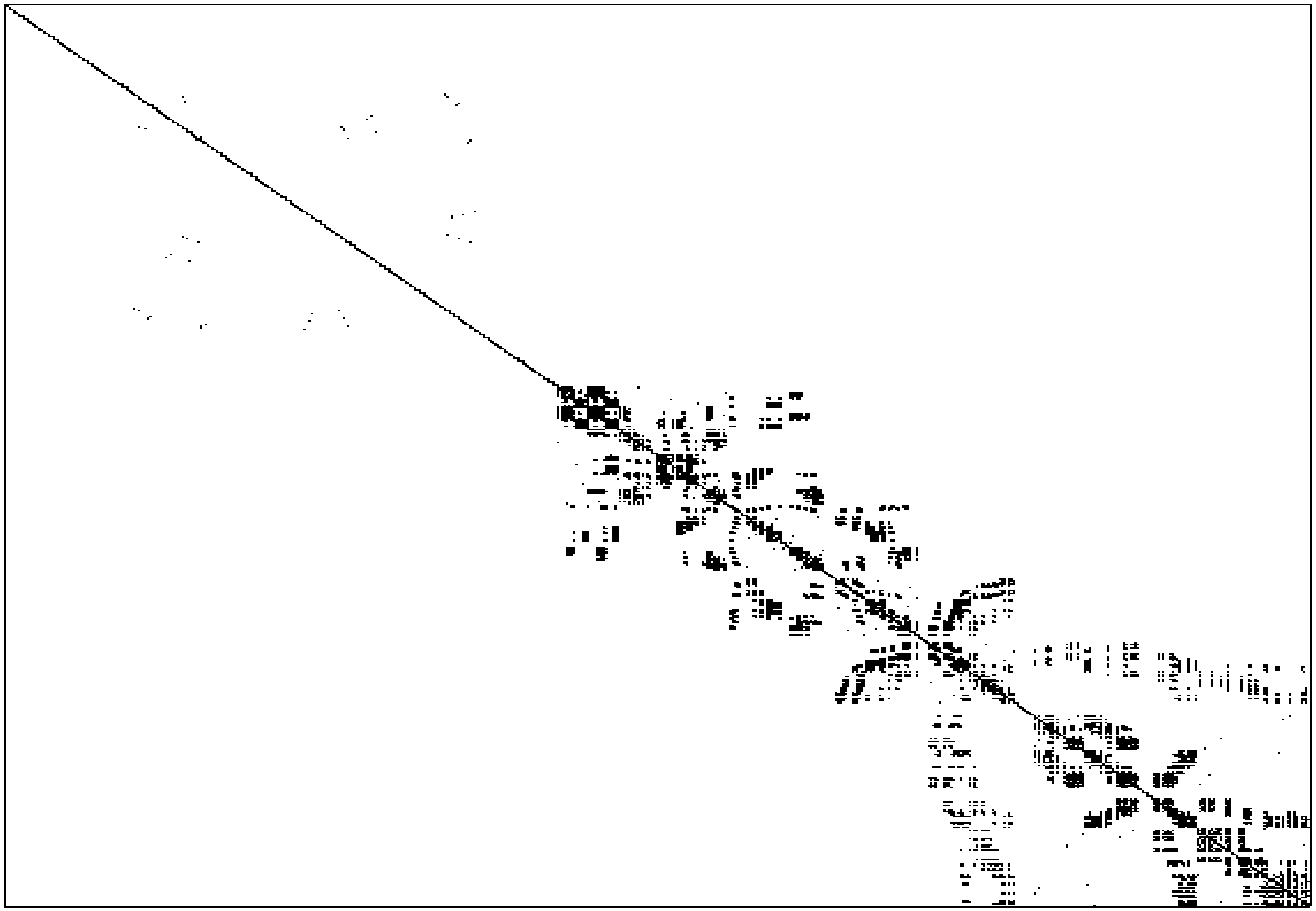


Thermal Simulation; SHERMAN2

Weather Matrix

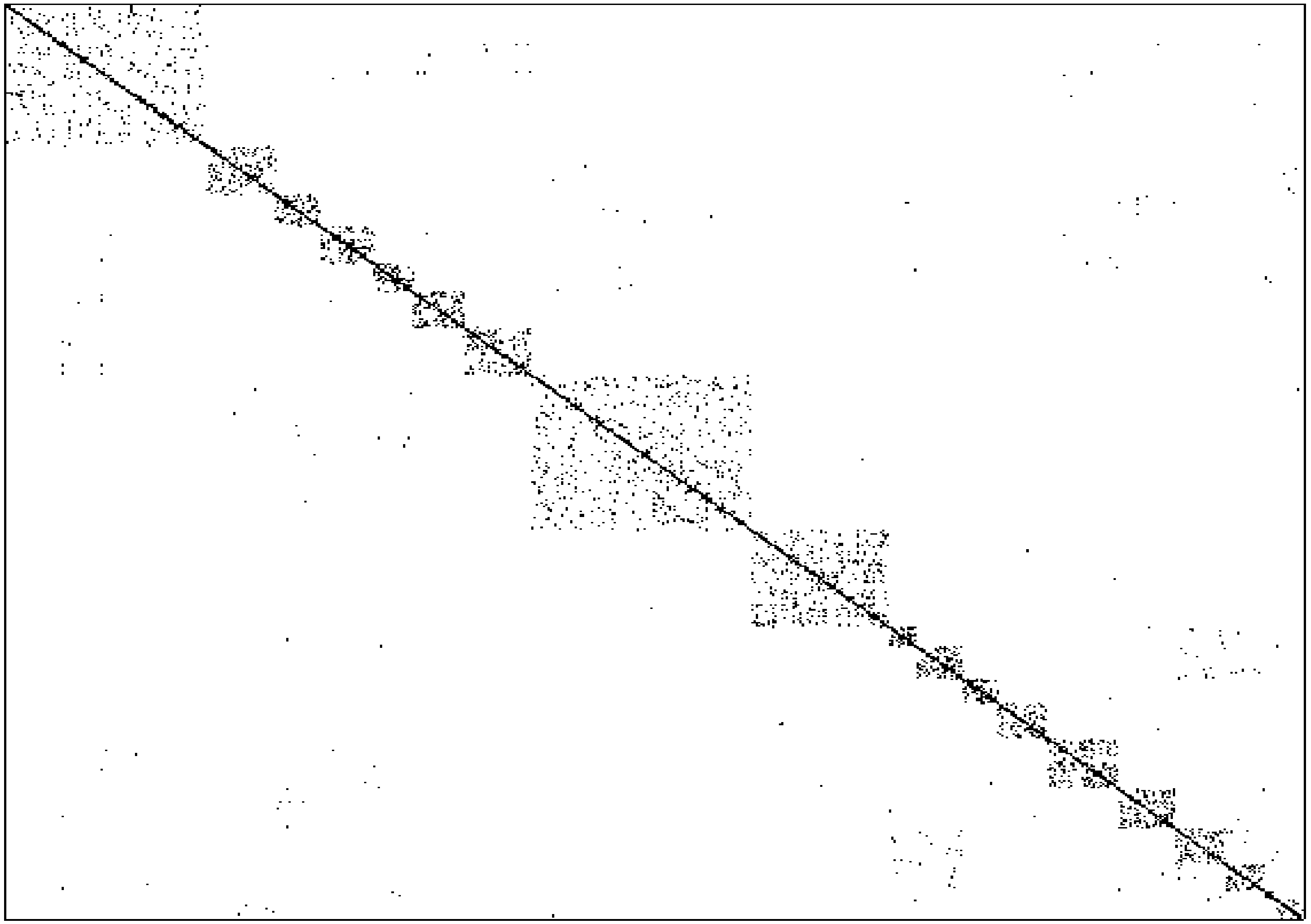

Weather Matrix; FS 760 3

Dynamic Calculation in Structures



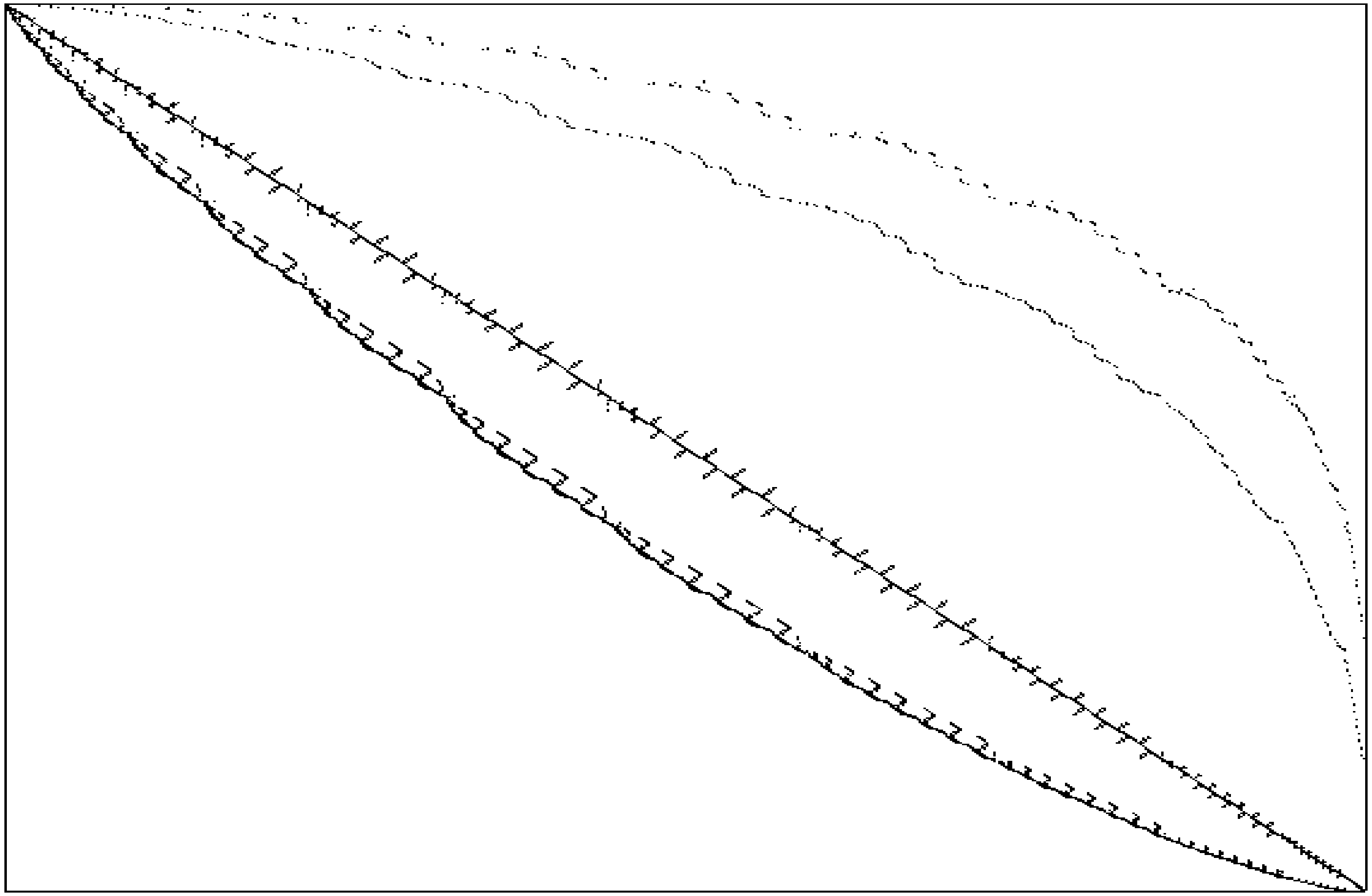
Dynamic Calculation in Structures; BCSSTM13

Power Systems



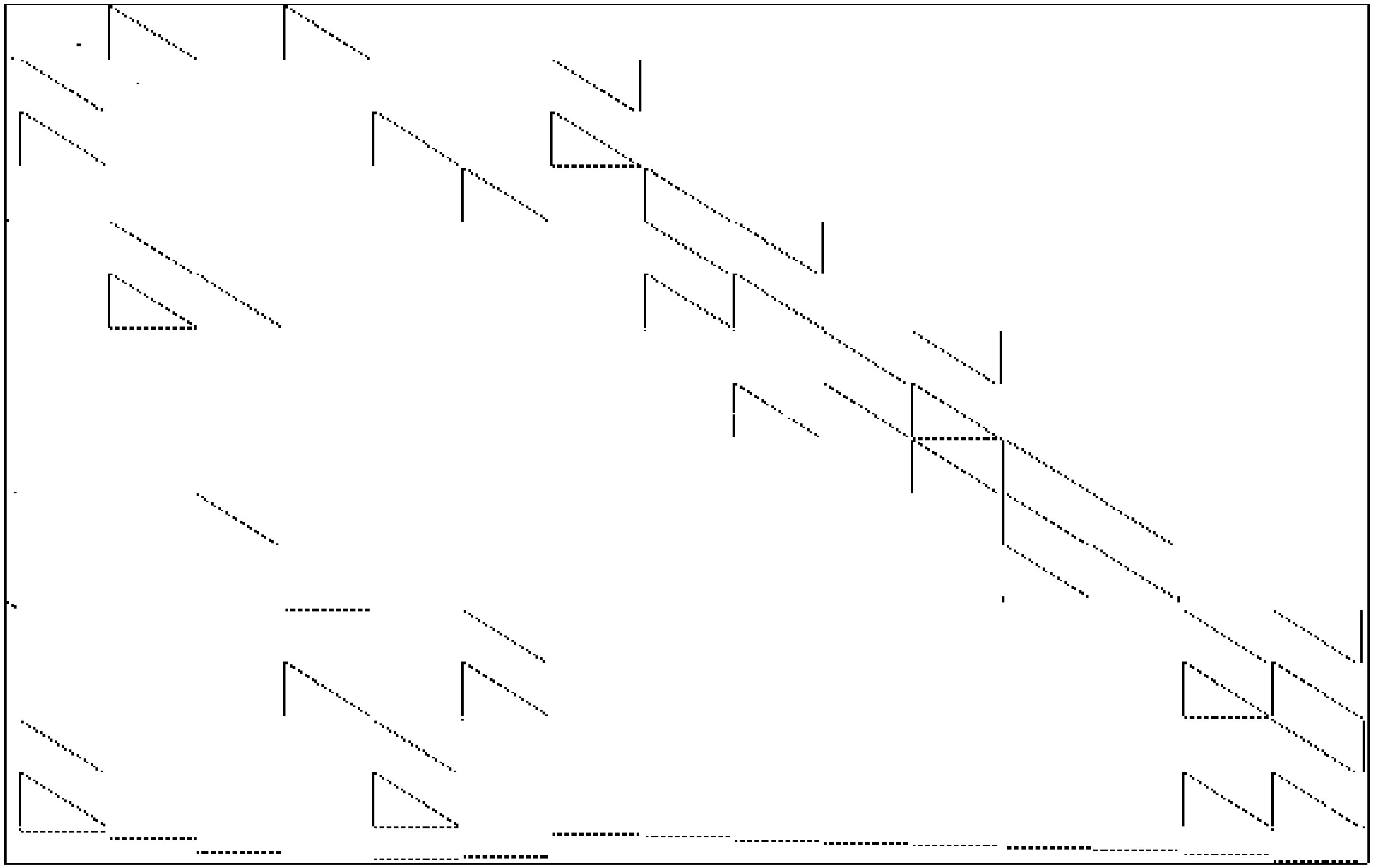
Power Systems; BCSPWR07

Simulation Of Computing Systems



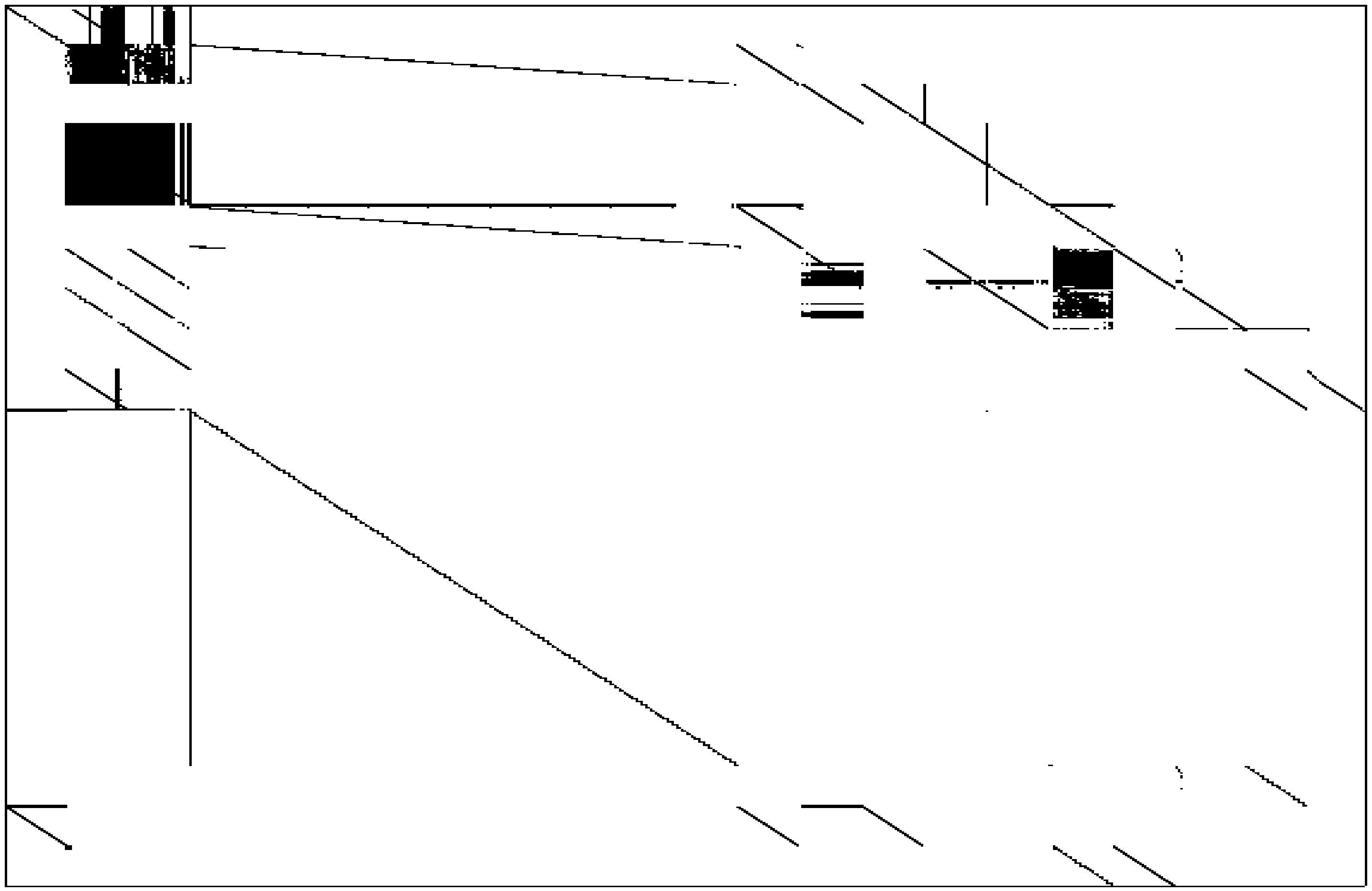
Simulation of Computing Systems; GRE 1107

Chemical Engineering



Chemical Engineering; WEST0381

Economic Modeling



Economic Modelling; ORANI678