Direct solution methods for sparse matrices

Direct solution methods for sparse matrices

Solve $A\mathbf{x} = \mathbf{b}$, where $A(n \times n)$.
(1) Factorize $A = LU$, L lower-triangular, U upper-triangular.
(2) Solve LU**x** = **b** as follows:

(2.1) Solve $L\mathbf{z} = \mathbf{b}$, i.e., $z_i = \dfrac{b_i - \sum\limits_{j=1}^{i-1} \ell_{i,j} z_j}{\ell_{i,i}}, \quad i = 1, 2, \cdots, n$

(2.2) Solve $U\mathbf{x} = \mathbf{z}$, i.e., $x_i = \dfrac{z_i - \sum\limits_{j=n}^{n-i} u_{i,j} x_j}{u_{i,i}}, \quad i = n, n-1, \cdots, 1$

Direct methods for sparse matrices

As the title indicates, we will analyse the process of triangular factorization (Gaussian elimination) and solution of systems with triangular matrices for the case of *sparse* matrices.
The direct solution procedure consists of factorization step and two triangular solves (forward and backward substitution).
Note: In general, during factorization we have to do pivoting in order to assure numerical stability.
The computational complexity of a direct solution algorithm is as follows.

Type of matrix A	Factor	LU solve	Memory
general dense	$2/3n^3$	$O(n^2)$	$n(n+1)$
symmetric dense	$1/3n^3$	$O(n^2)$	$1/2n(n+1)$
band matrix $(2q+1)$	$O(q^2n)$	$O(qn)$	$n(2q+1)$

What is a sparse matrix? - $nnz(A) = O(N), A(N \times N)$.

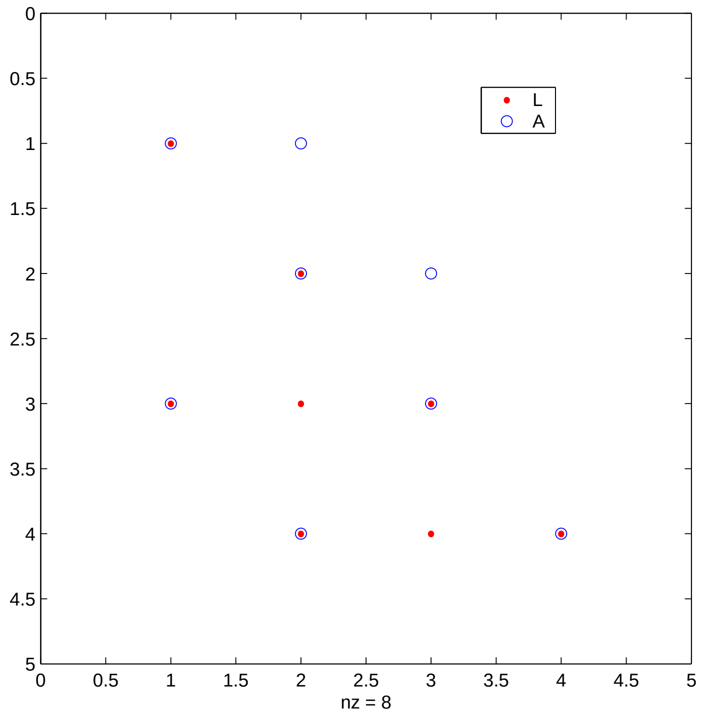
Let us see some sparse matrices.

The following slides are borrowed from Iain Duff.

Special thanks.

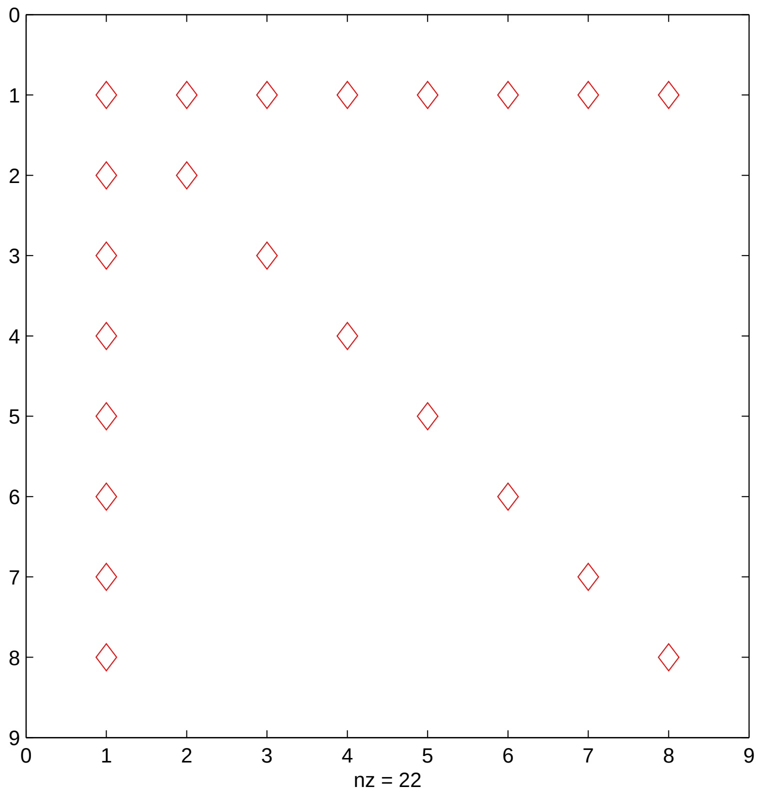
Why are we concerned separately with direct methods for sparse matrices?

The reason to consider especially factorizations of sparse matrices is the effect of *fill-in*, namely, obtaining nonzero entries in the LU factors in positions where $A_{i,j}$ is zero. This is easy to be seen from the basic Gaussian elimination operation:

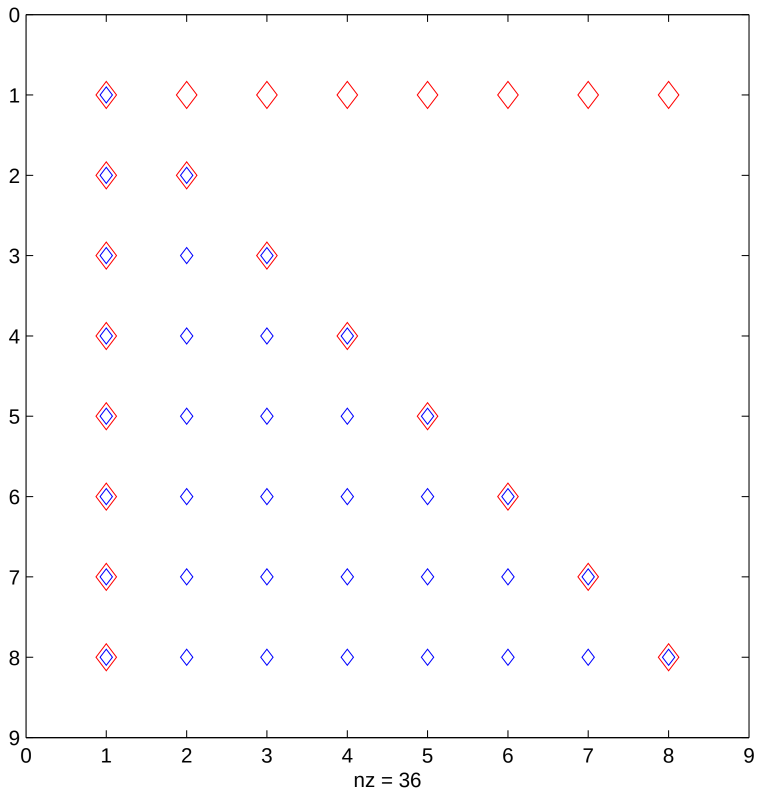


$$a_{i,j}^{(k+1)} \longleftarrow a_{i,j}^{(k)} + \frac{a_{i,k}^{(k)}\ a_{k,j}^{(k)}}{a_{k,k}^{(k)}}$$

Concerns during the factorization phase:

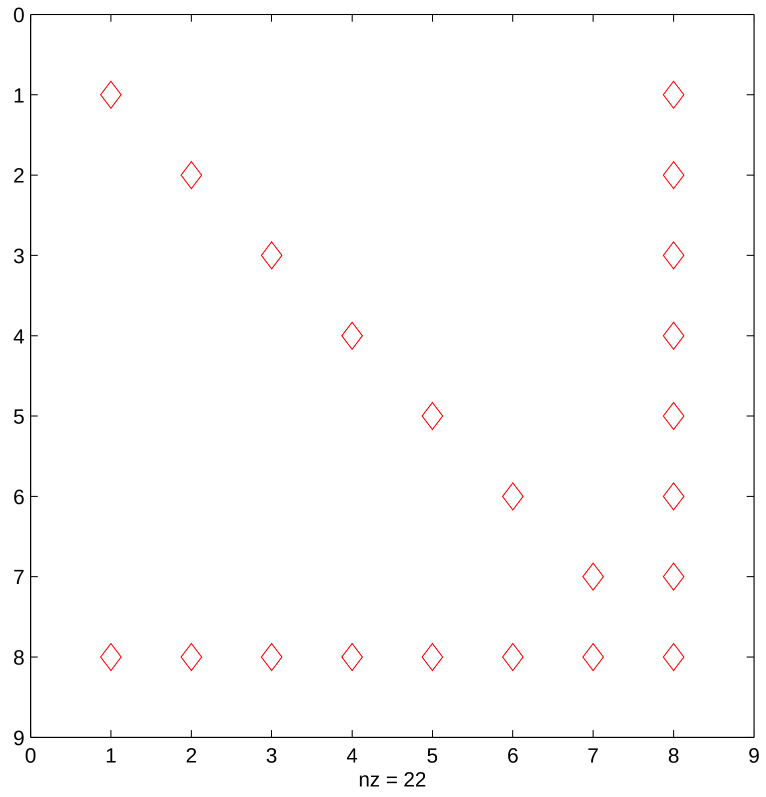


(a) Arrow matrix

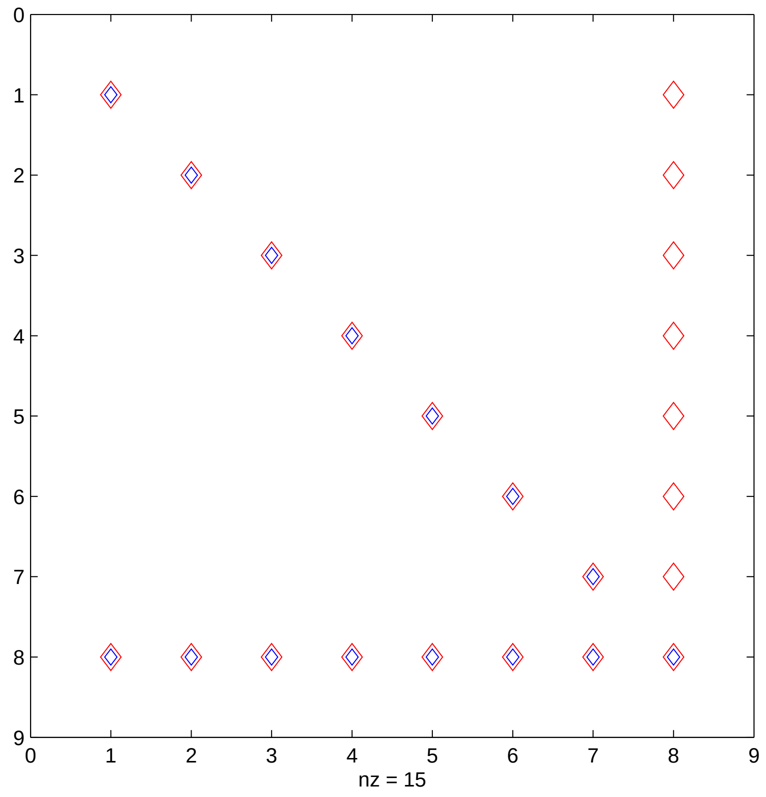


(b) The structure of the L-factor

The arrow matrix structure - the L and U factors are full.



(c) Arrow matrix permuted



(d) The structure of the L-factor

We can permute the matrix A first and then factorize!

We pose now the the question to find permutation matrices P and Q, such that when we factorize $\widetilde{A} = Q^T A P^T$, the fill-in in the then obtained L and U factors will be minimal.
The solution algorithm takes the form:
(1) Factorize $Q^T A P^T = LU$ (2) Solve $PL\mathbf{z} = \mathbf{b}$ and $UQ\mathbf{x} = \mathbf{z}$.
How to construct P and Q in general?

Two possible situations will be considered:

(M) We are given only the matrix, thus we can utilize only the structure of A (*matrix-given strategies*);

(P) We know the origin of the sparse linear system and we are permitted to use this knowledge to construct A so that it has a favourable structure (*problem-given strategies*).

Along the road, we will also briefly discuss the suitability of the approaches for parallel implementation on HPC/parallel computers.

The aim of sparse matrix algorithms is to solve the system $A\mathbf{x} = \mathbf{b}$ in time and space (computer memory requirements) proportional to $\boxed{O(n) + O(nnz(A)),}$ where $nnz(A)$ denotes the number of nonzero elements in A.

Even if the latter target cannot be achieved, the complexity of sparse linear algebra is far less than that of the dense case:

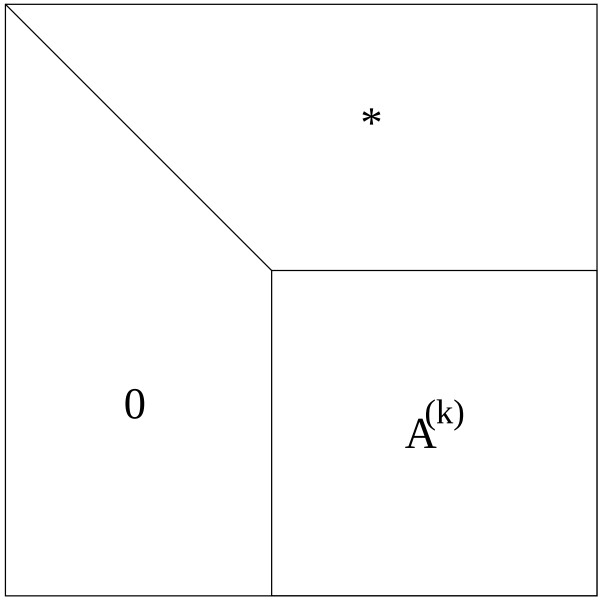
Order		Time in sec	
of A	$nnz(A)$	Dense solver	Sparse solver
680	2646	0.96	0.06
1374	8606	6.19	0.70
2205	14133	24.25	2.65
2529	90158	36.37	1.17

Time on Cray Y-MP (results taken from I. Duff)

The strive to achieve complexity $O(n) + O(nnz(A))$ entails very complicated sparse codes. We name some aspects which fall out of the scope of the present course but play and important role when implementing the direct solution techniques for sparse matrices in practice.

- sparse data structures and manipulations with those;
- computer platform related issues, such as handling of indirect addressing; lack of locality; difficulties with cache-based computers and parallel platforms; short inner-most loops;

Extra difficulties come from the fact that we have to choose a pivot element and its proper choice may contradict to the strive to minimize fill-in.
As an illustration, we consider the following strategy for maintaining sparsity (due to Markowitz, 1957). Consider the k-th step of the Gaussian elimination:



$$A^{(k)} = \begin{matrix} a_{k,k}^{(k)} & \cdots & a_{k,j}^{(k)} & \cdots & a_{k,n}^{(k)} \\ \vdots & & & & \vdots \\ a_{i,k}^{(k)} & \cdots & a_{i,j}^{(k)} & \cdots & a_{i,n}^{(k)} \\ \vdots & & & & \vdots \\ a_{n,k}^{(k)} & \cdots & a_{n,j}^{(k)} & \cdots & a_{n,n}^{(k)} \end{matrix}$$

$A^{(k)}$ is of order $n-k+1$. Let $n_i^{(k)}$ and $n_j^{(k)}$ be the number of nonzero entries in the ith row and the jth column of $A^{(k)}$, respectively. Choose pivot $a_{i,j}^{(k)}$ such that the expression $(n_i^{(k)}-1)(n_j^{(k)}-1)$ is minimized.

Condition $min[(n_i^{(k)} - 1)(n_j^{(k)} - 1)]$ can be seen as

- choosing a pivot which will modify the least number of coefficients in the remaining submatrix;
- choosing a pivot that involves least multiplications and divisions;
- as a means to limit the fill-in since it will produce at most $(n_i^{(k)} - 1)(n_j^{(k)} - 1)$ new nonzero entries.

However, in general the entry $a_{i,j}^{(k)}$ has to obey some other numerical criteria also, for example,

$$|a_{i,j}^{(k)}| \geq \tau |a_{i,s}^{(k)}|, i \geq s,$$

where $\tau \in (0, 1)$ is a threshold parameter.

The following table illustrates how the choice of τ can influence the stability of the factorization.

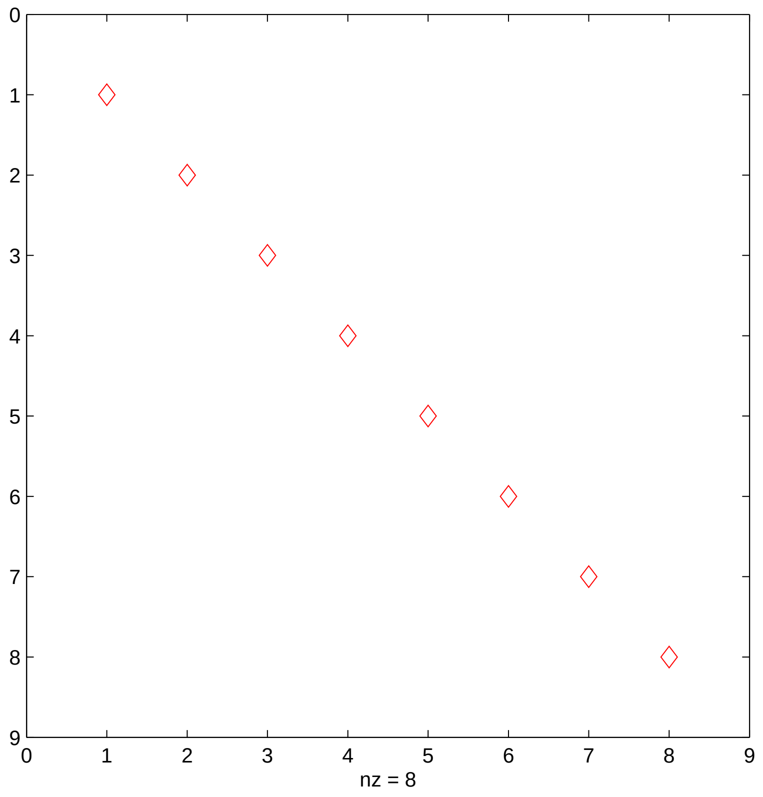
τ	$nnz(L, U)$	Error in solution
1.0	16767	3e-09
0.25	14249	6e-10
0.10	13660	4e-09
0.01	15045	1e-05
1e-4	16198	1e+02
1e-10	16553	3e+23

"Given-the-matrix" strategy

Given-the-matrix *strategy*

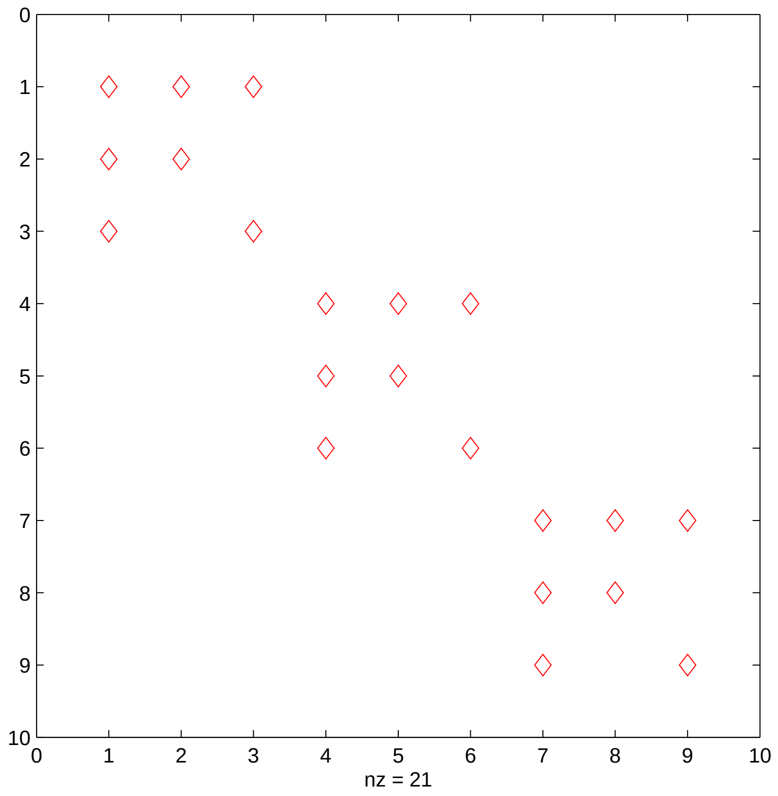
In the *given-the-matrix* case the only source of information is the matrix itself and we will try to reorder the entries so that the resulting structure will limit the possible fill-in.
What is the matrix structure to aim at?

Given-the-matrix *strategy*

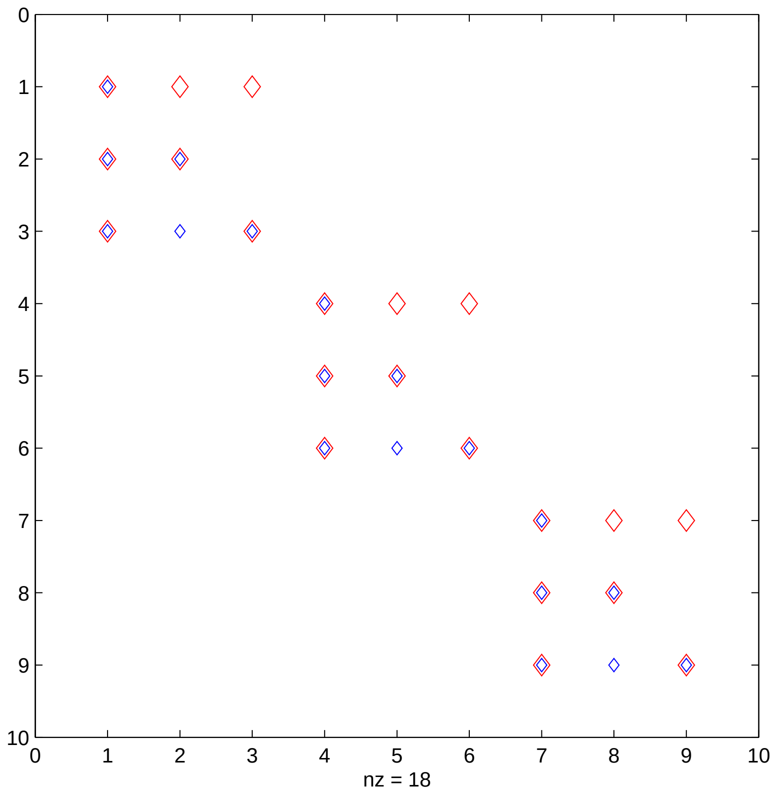


(e) Diagonal matrix

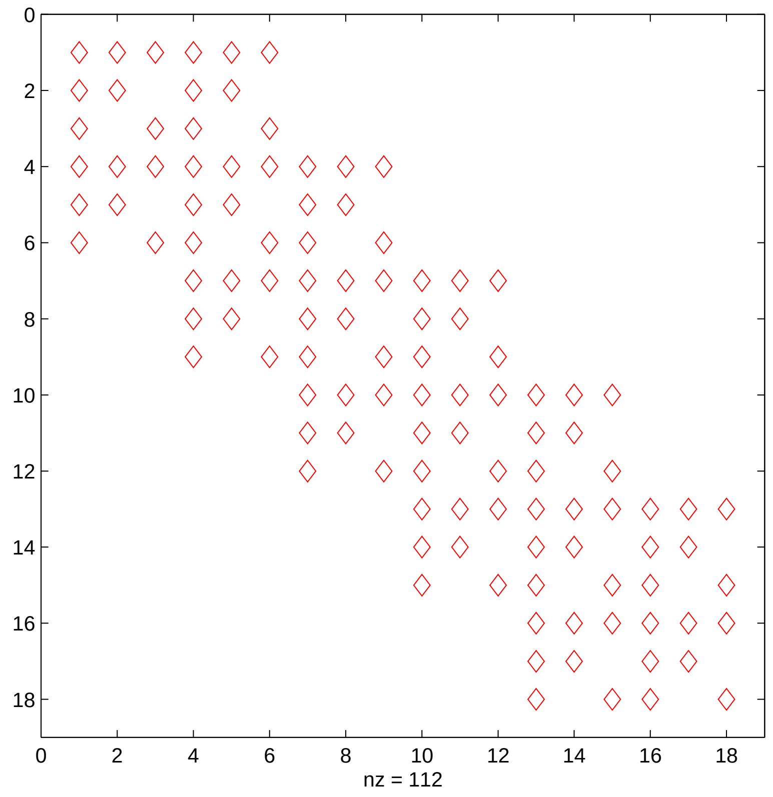
- diagonal
- block-diagonal
- block-tridiagonal
- arrow matrix
- band matrix
- block-triangular

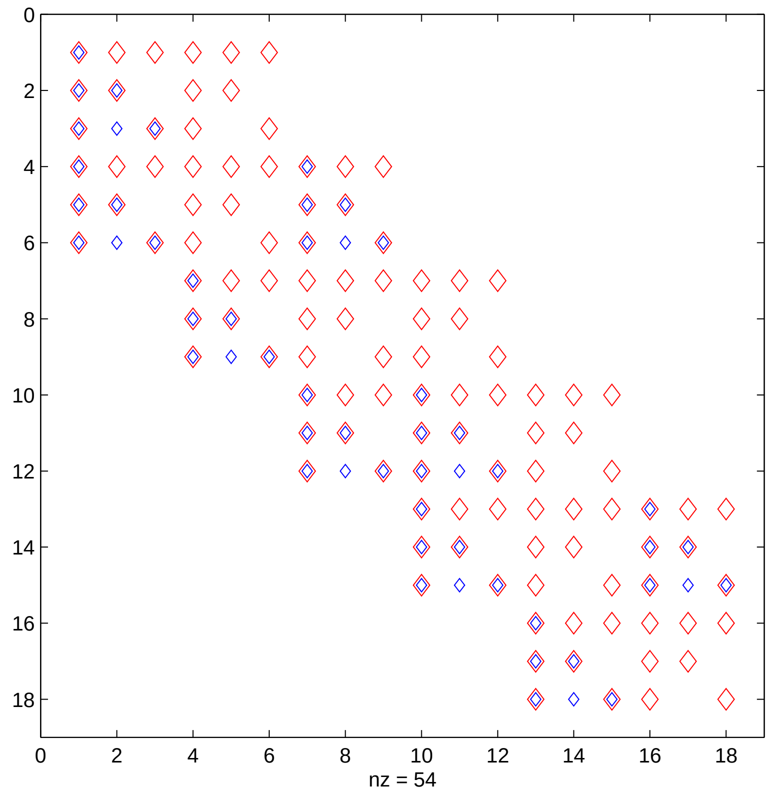


(f) block-diagonal matrix



(g) The structure of the L-factor


(h) Block-tridiagonal matrix


(i) The structure of the L-factor

We will consider the case of symmetric matrices ($P = Q$) and three popular methods based on manipulations on the graph representation of the matrix.
- (generalized) reverse Cuthill-McKee algorithm (1969);
- nested dissection method (1973);
- minimum degree ordering (George and Liu, 1981) and variants.

Some common graph notations:

Given a graph $G(A) = (V, E)$ of a symmetric matrix A, where V is a set of vertices, E - a set of edges.

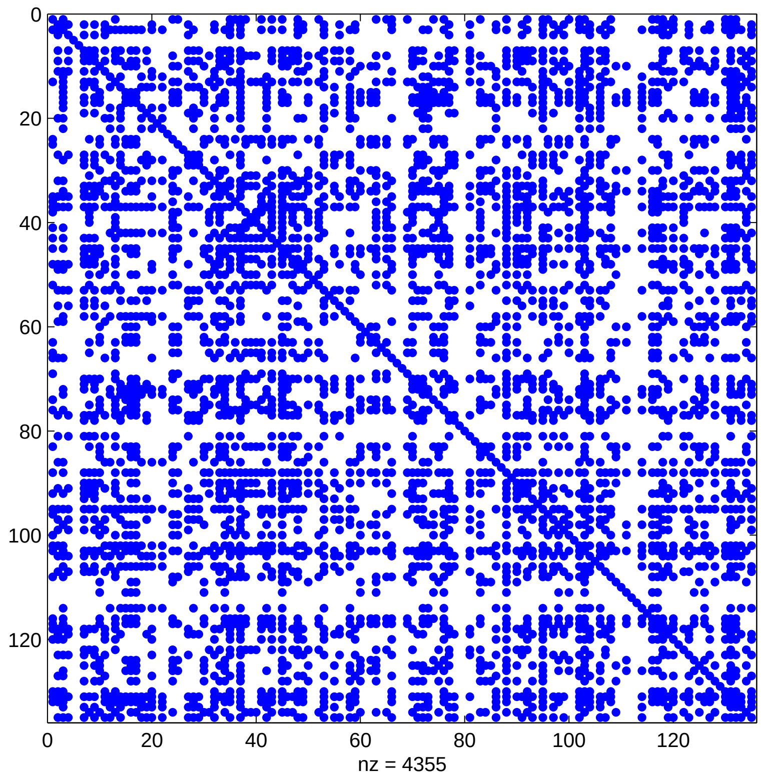
– A pair of vertices $(v_i, v_j), \quad i \neq j$, is an edge in $G(A)$ if and only if $a_{ij} \neq 0$.

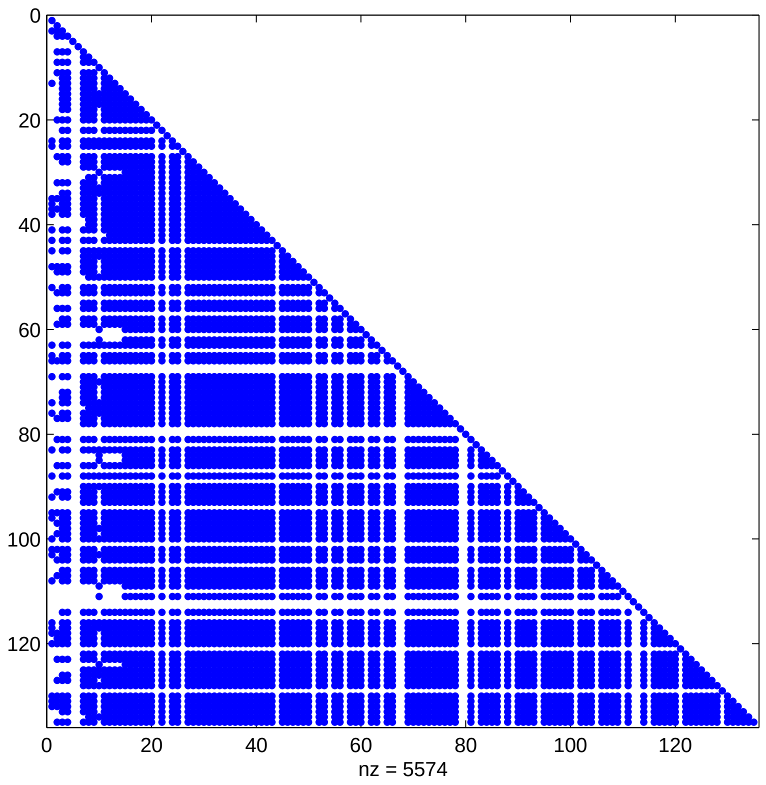
– Two vertices v, w called adjoint, if $(v, w) \in E$.

– If $W \in V$ is a given set of vertices of the graph $G(A)$, then an adjoint set for W (with respect to V) is $Adj(W) = \{v \in V - W$ such that $\{v, w\} \in E$ for some $w \in W\}$.

– A degree $|W|$ of $W \in V$ is the number of elements in $Adj(W)$. In particular, the degree of a vertex w is defined as a number of vertices adjoint to w.

A matrix from somewhere





Generalized Reverse Cuthill-McKee alg.(RCM)

Aim: minimize the envelope (in other words a band of variable width) of the permuted matrix.

1. *Initialization*. Choose a starting (root) vertex r and set $v_1 = r$.

2. *Main loop*. For $i = 1, ..., n$ find all non-numbered neighbours of v_i and number them in the increasing order of their degrees.

3. *Reverse order*. The reverse Cuthill-McKee ordering is $w_1, ..., w_n$, where $w_i = v_{n+1-i}$.

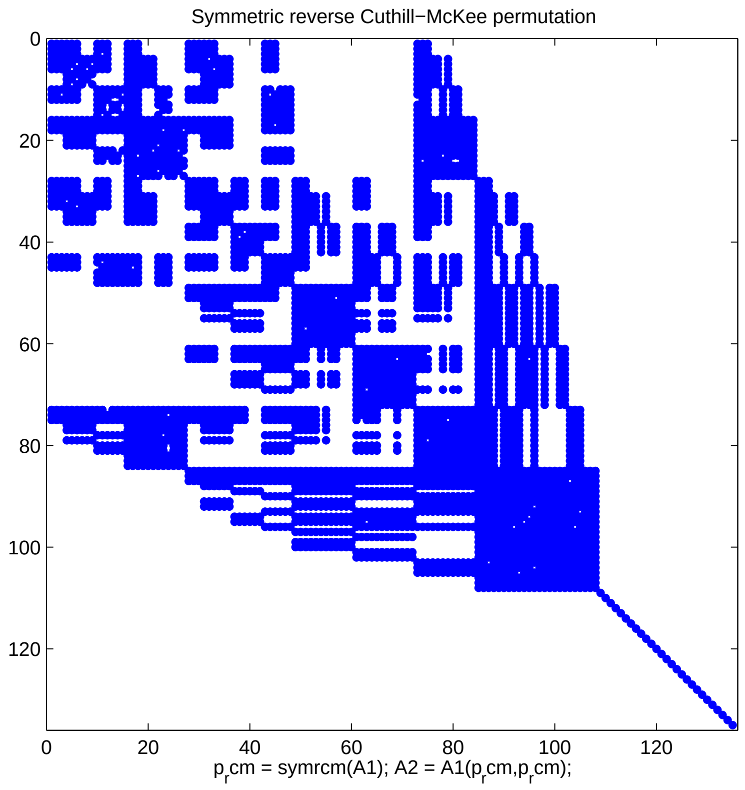
Generalized Reverse Cuthill-McKee alg.(RCM)

One can see that GenRCM tends to number first the vertices adjoint to the already ordered ones, i.e., it gathers matrix entries along the main diagonal.

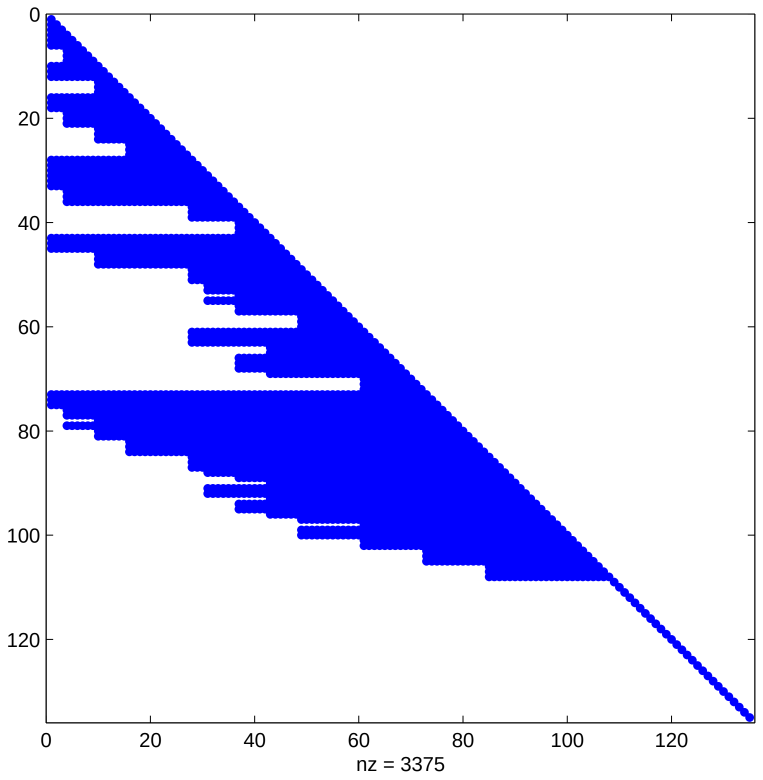
The choice of a root vertex is of a special interest.

The complexity of the algorithm is bounded from above by $O(m\ nnz(A))$, where m is a maximum degree of vertices, $nnz(A)$ - number of nonzero entries of matrix A.

Generalized Reverse Cuthill-McKee alg.(RCM)



p_rcm = symrcm(A1); A2 = A1(p_rcm,p_rcm);

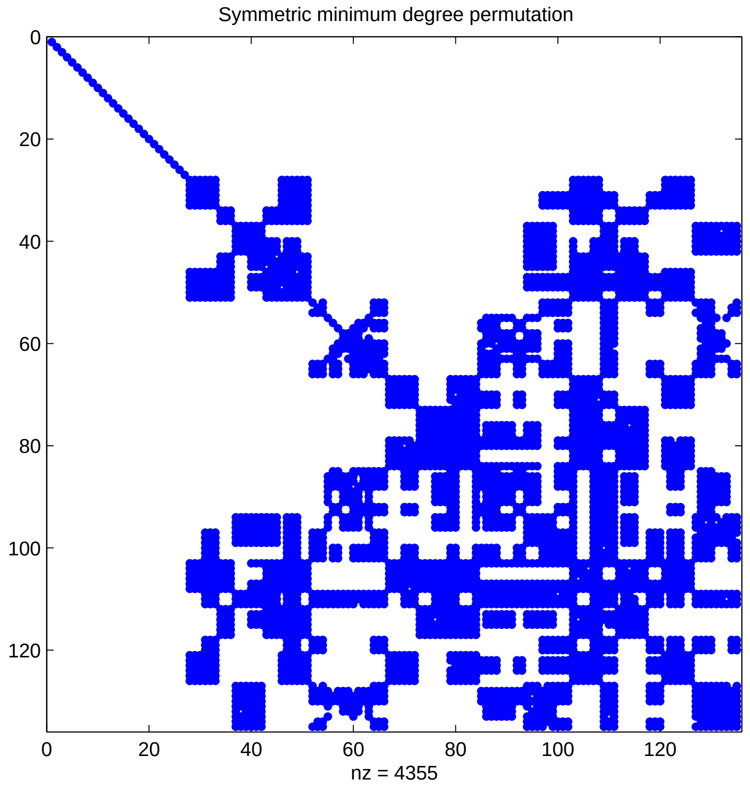


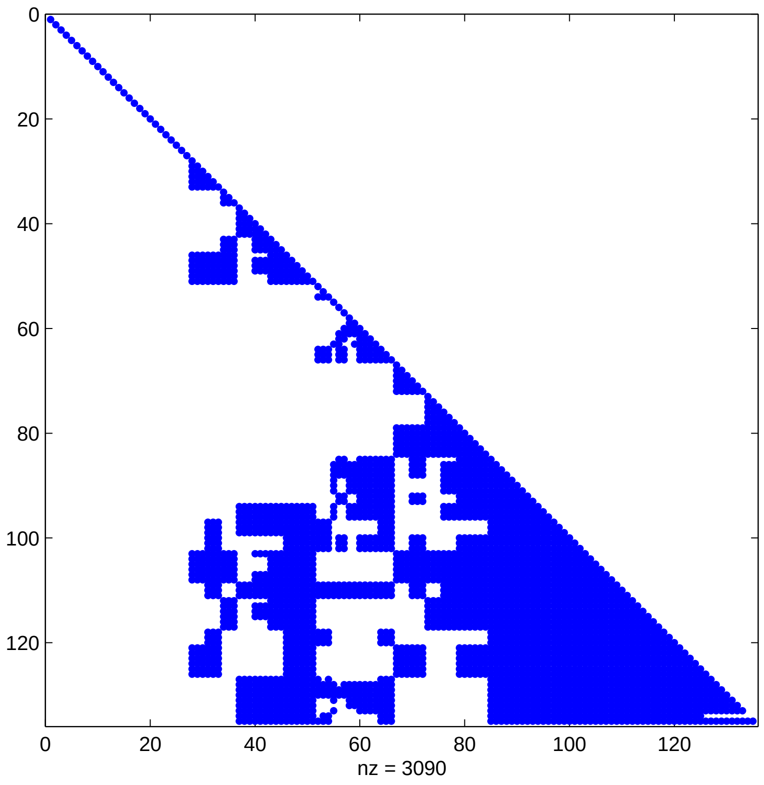
The Quotient Minimum Degree (QMD)

Aims to minimize a local fill-in taking a vertex of minimum degree at each elimination step. The straightforward implementation of the algorithm is time consuming since the degree of numerous vertices adjoint to the eliminated one must be recomputed at each step. Many important modifications have been made in order to improve the performance of the MD algorithm and this research remains still active .

In many references the MD algorithm is recommended as a general purpose fill-reducing reordering scheme. Its wide acceptance is largely due to its effectiveness in reducing fill and its efficient implementation.

The Quotient Minimum Degree (QMD)





The Nested Dissection algorithm

A recursive algorithm which on each step finds a separator of each connected graph component. A separator is a subset of vertices whose removal subdivides the graph into two or more components. Several strategies how to determine a separator in a graph are known. Numbering the vertices of the separator last results in the following structure of the permuted matrix with prescribed zero blocks in positions $(2,1)$ and $(1,2)$

$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

The Nested Dissection algorithm

Under the assumption that subdivided components are of equal size the algorithm requires no more than $\boxed{\log_2 n}$ steps to terminate.

ND is optimal (up to a constant factor) for the class of model 9-point two-dimensional grid problems posed on regular $m \times m$-meshes. In this case a direct solver based on the ND ordering requires $O(m^3)$ arithmetic operations for matrix factorization and $O(m^2 log_2 m)$ arithmetic operations to solve triangular systems. Accordingly, the Cholesky factor contains $O(m^2 log_2 m)$ nonzero entries. These are the <u>*best low order bounds*</u> derived for direct elimination methods. In the three dimensional case and model 27-point grid problems on cubic $m \times m \times m$ meshes the number of factorization operations is estimated as $O(m^6)$.
Therefore one can expect iterative methods to be in general superior for 3D grid problems and for large enough 2D problems. This holds in particular for reasonably well-conditioned problems and not too irregular grids.

A comparison...

Method	Direct			
	RCM	ND	QMD	ND*
Time (sec)	45.82	39.54	171.84	783.88

Comparison results, problem size $n = 92862$

Direct methods, 2D problem: Bridge

Method	Ordering	Factorization	Solution	Total time	True res.
$n = 6270$ $nzA = 80726$					
RCM	0.0321	0.3311	0.0356	0.3987	1.81-09
ND	0.1270	0.5167	0.0390	0.6826	1.33-09
QMD	0.6852	0.3735	0.0350	1.0937	1.30-09
$n = 23838$ $nzA = 316752$					
RCM	0.1440	3.4762	0.2550	3.875	1.72-09
ND	0.6476	4.0399	0.2062	4.894	1.25-09
QMD	9.1588	3.5092	0.1952	12.863	1.22-09
$n = 92862$ $nzA = 1254552$					
RCM	0.601	43.306	1.908	45.82	9.25-09
ND	3.296	35.139	1.109	39.54	5.73-09
QMD	138.65	32.046	1.139	171.84	6.15-09
$n = 366462$ $nzA = 4993118$					
RCM	2.552	1100.7	25.01	1127.7	5.18-08
ND	15.86	320.8	5.43	342.1	2.41-08
QMD	2168.6	410.8	6,23	2585.6	2.66-08

Direct methods, 2D problem: Dam

Method	Ordering	Factorization	Solution	Total time	True res.
$n = 13474$ $nzA = 182502$					
RCM	0.0688	3.0018	0.1853	3.256	2.58-12
ND	0.3141	3.4374	0.1314	3.883	1.32-12
QMD	2.5228	2.6335	0.1237	5.280	1.30-12
$n = 53058$ $nzA = 729582$					
RCM	0.3179	47.014	1.4091	48.841	1.30-11
ND	1.7335	31.600	0.6948	34.028	5.17-12
QMD	40.9980	31.200	0.7148	72.913	5.51-12
$n = 210562$ $nzA = 2917310$					
RCM	1.41	1303.30	11.366	1316.0	6.32-11
ND	9.41	300.10	3.558	313.1	1.91-11
QMD	777.35	310.97	3.838	1091.2	2.08-11
$n = 838914$ $nzA = 11667410$					
RCM	5.442	out of	memory	-	-
ND	41.48	2696.36	17.16	2755.0	6.66-11
QMD	12751.00	3819.30	40.97	16612.0	7.54-11

Direct methods, 3D problem: Bricks

Method	Ordering	Factorization	Solution	Total time	True res.
$n = 135 \quad nzA = 4313$					
RCM	0.0017	0.0040	0.0003	0.0059	2.95-14
ND	0.0019	0.0062	0.0005	0.0086	1.14-14
QMD	0.0109	0.0056	0.0004	0.0170	1.62-14
$n = 675 \quad nzA = 32817$					
RCM	0.0119	0.1042	0.0031	0.119	1.05-13
ND	0.0190	0.2303	0.0060	0.255	4.72-14
QMD	0.1552	0.2560	0.0063	0.417	5.92-14
$n = 4131 \quad nzA = 255515$					
RCM	0.0987	9.1881	0.1085	9.40	4.85-13
ND	0.2252	16.892	0.1645	17.28	2.55-13
QMD	1.9759	25.543	0.1991	27.72	2.74-13
$n = 28611 \quad nzA = 2016125$					
RCM	0.821	1189.4	4.903	1195.1	3.85-12
ND	1.907	650.9	2.134	654.9	1.05-12
QMD	35.654	3537.8	5.607	3579.1	1.55-12

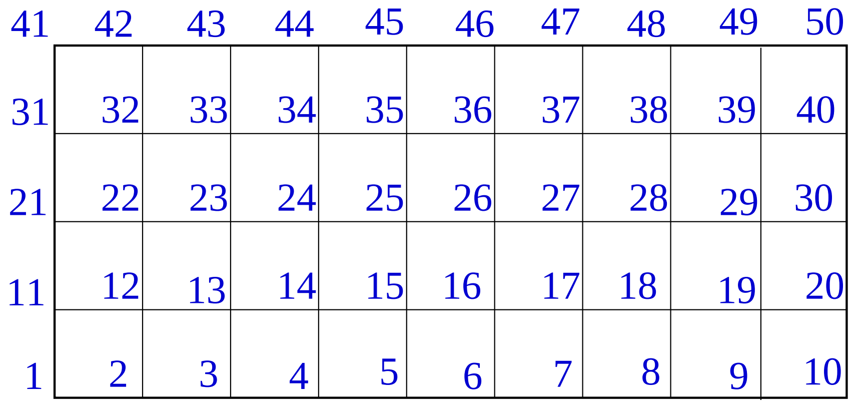
Direct methods, 3D problem:Soil

Method	Ordering	Factorization	Solution	Total time	True res.
$n = 375 \quad nzA = 15029$					
RCM	0.0040	0.0297	0.0013	0.0350	9.03-15
ND	0.0071	0.0544	0.0021	0.0636	6.29-15
QMD	0.0475	0.0574	0.0022	0.1070	6.06-15
$n = 2187 \quad nzA = 122441$					
RCM	0.0337	4.2352	0.0599	4.329	4.77-14
ND	0.0901	3.8116	0.0619	3.964	2.55-14
QMD	0.5192	3.3087	0.0466	3.875	2.28-14
$n = 14739 \quad nzA = 500688$					
RCM	0.3280	672.82	1.918	675.07	3.69-13
ND	1.3481	243.76	1.110	246.21	1.03-13
QMD	8.5856	707.95	1.549	718.09	1.43-13
$n = 107811 \quad nzA = 7925773$					
RCM	2.643	out of	memory	-	-
ND	15.627	18420.3	27.695	18464.0	4.67-13
QMD	319.297	out of	memory	-	-

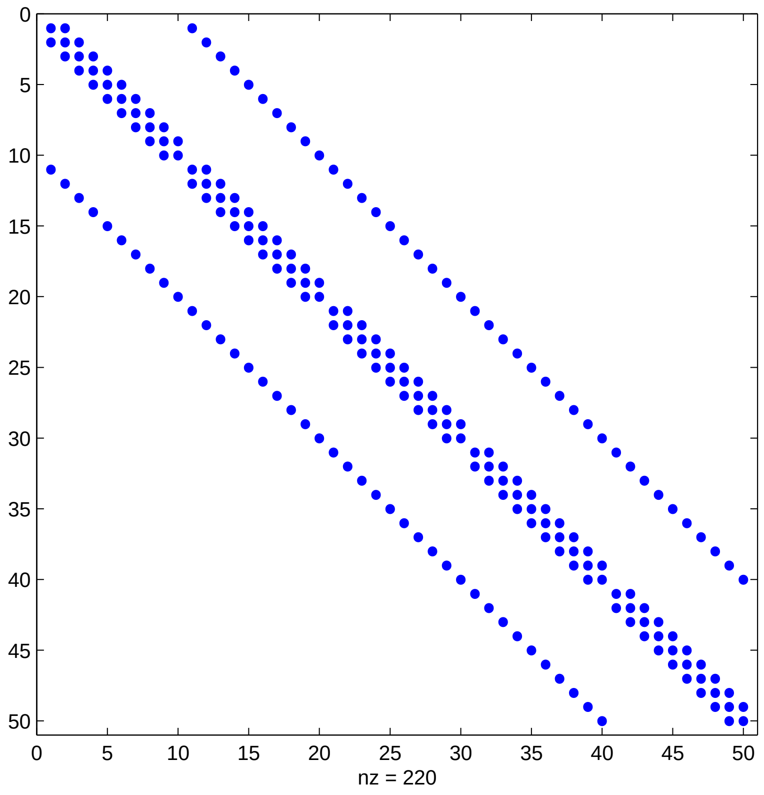
"Given-the-problem" strategy

Given-the-problem *strategy*

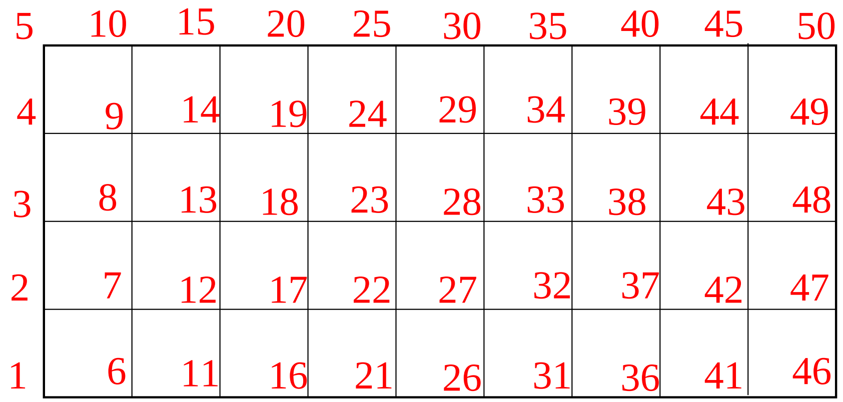
Assume <u>we know the origin of the linear system</u> of equations to be solved. In many cases it comes from a numerically discretized (system of) PDEs, and we know the domain of definition of the problem (Ω), its geomethical properties, the discretization method (finite differences (FD), finite elements (FE), finite volumes (FV), boundary integral (BE) method). In such cases the system matrix enjoys a special structure. This information can be utilized while computing the matrix so that it will be constructed in (almost) favourable form.



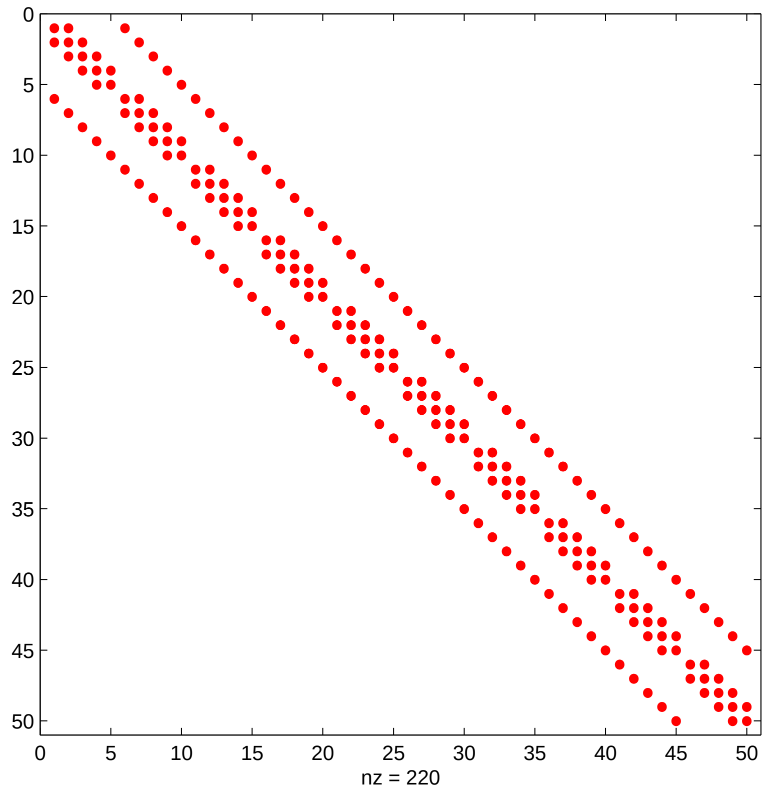
(j) Row-wise ordering



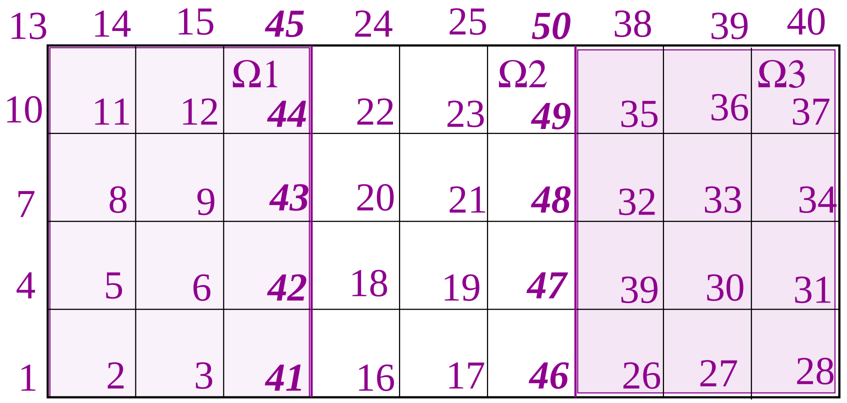
(k) The structure of the matrix A



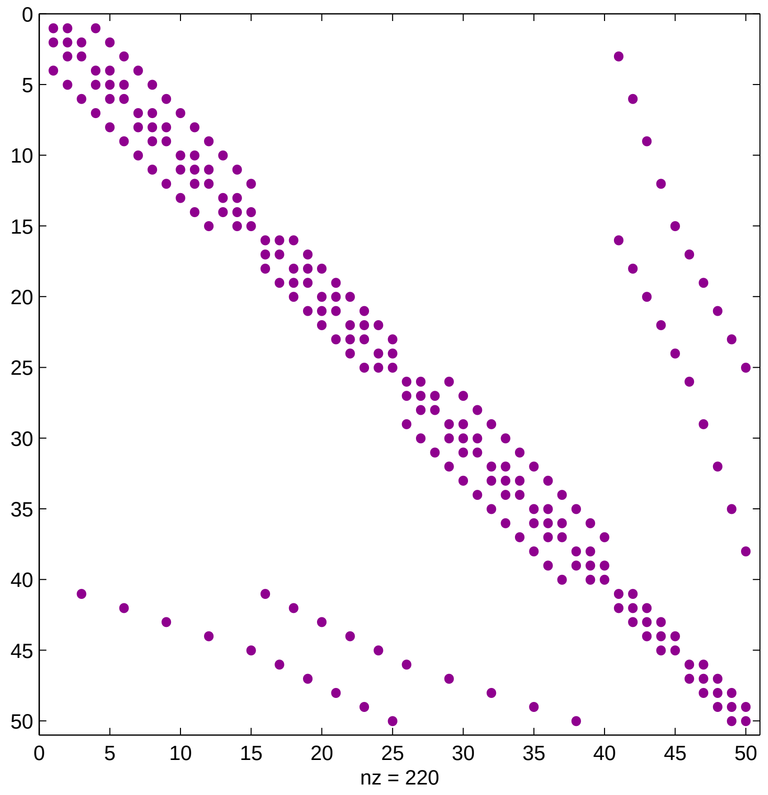
(l) Column-wise ordering



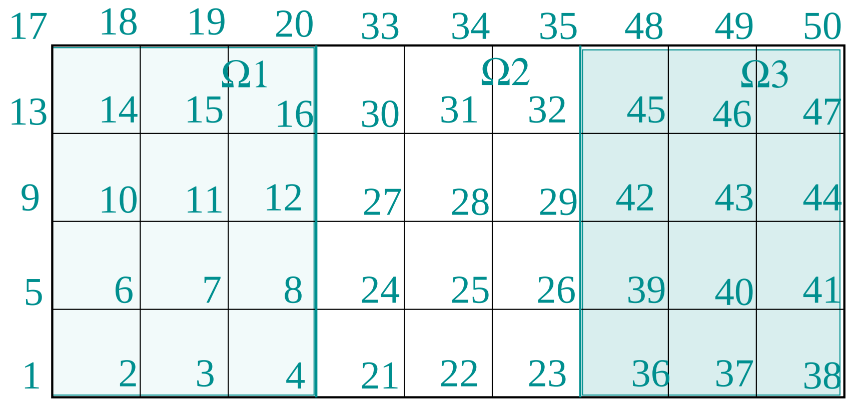
(m) The structure of the matrix A



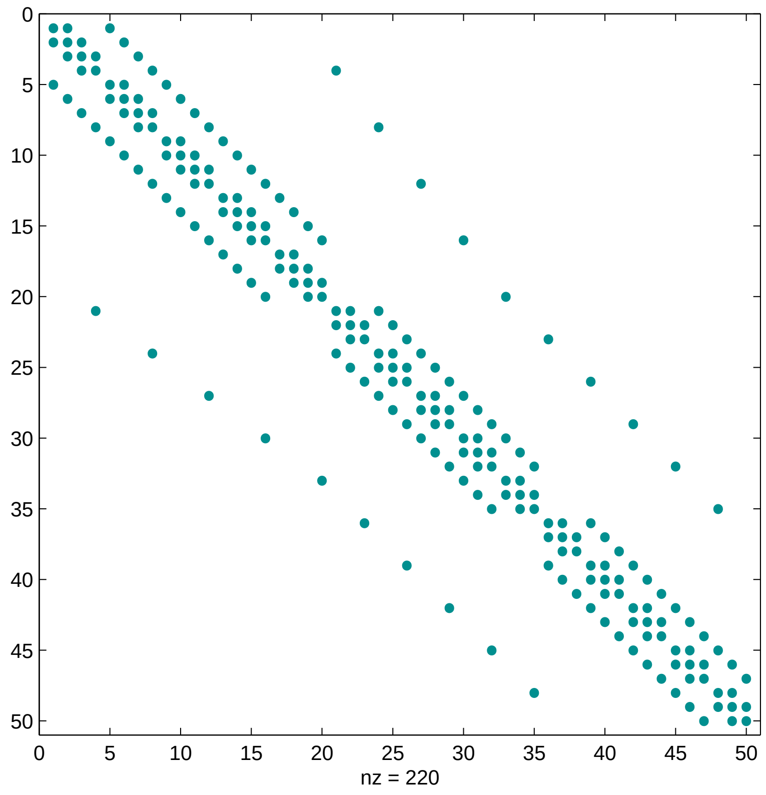
(n) Domain decomposition ordering 1



(o) The structure of the matrix A



(p) Domain decomposition ordering 2



(q) The structure of the matrix A

Edge-research in direct solution methods for sparse matrices

• Ordering techniques to singly bordered block-diagonal forms for unsymmetric parallel direct solvers

$$\begin{bmatrix} A_{11} & & & & B_1 \\ & A_{22} & & & B_2 \\ & & \dots & & \vdots \\ & & & A_{nn} & B_n \end{bmatrix}$$

• MUMPS, MUltifrontal Massively Parallel Solver: an international project to design and support a package for the solution of large sparse systems using a multifrontal method on distributed memory machines.

Borowed from Iain Duff

AUDI-CRANKSHAFT

Order:	943695
Nonzero entries:	39 297 771
Analysis	
Entries in factors:	1 435 757 859
	11.2 GBytes
Operations required:	$5.9\,10^{12}$
Factorization (SGI ORIGIN at Bergen)	
1 Processor:	32000 sec
	16 GBytes
2 Processors:	22000 sec
	20 GBytes

	Assembly time (s)		Total solution time (s)		
N	Abaqus	Iterative	Abaqus	Iterative	
				time	iterations
		2D			
6043	1	0.2178	1.098	1.02 (0.4863)	13 (1,1)
23603	3.326	0.8857	4.718	4.225 (1.995)	12 (1,1)
93283	13.02	3.978	18.05	19.38 (9.813)	11 (2,1)
370883	50.54	17.71	72.98	89.34 (49.43)	11 (2,1)
1479043	269.1	77.7	317.5	431.8 (257.6)	12 (2,1)
		3D			
12512	1.525	1.899	3.049	8.009 (3.465)	12 (2,1)
89700	14.09	8.756	43.29	63.34 (33.08)	13 (2,1)
678116	110.3	65.8	1347	749.3 (506.8)	15 (4,1)

Summary:

- There is no one good buy.
- The best code in any situation will depend on
  - the solution environment;
  - the computing platform;
  - the structure of the matrix.

Some references:

P. R. Amestoy, T. A. Davis and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matr. Anal. Appl.*, 17, 886-905, 1996.
C. Ashcraft and J.W.H. Liu. Robust ordering of sparse matrices using multisection. *SIAM J. Matrix Anal. Appl.*, 19, 816-832, 1998.
E. Cuthill, J. McKee. Reducing the bandwidth of sparse symmetric matrices. *Proc. 24th Nat. Conf. Assoc. Comput. Mach.*, 157-172, 1969.
J.W.H. Liu, A. H. Sherman. Comparative analysis of the Cuthill-McKee and the reverse Cuthill-Mckee ordering algorithms for sparse matrices. *SIAM J. Numer. Anal.*, 13, 198-213, 1975.
J. Dongarra, I. Duff, Sorensen and H. van der Vorst, *Numerical Linear Algebra for High Performance Computers*, SIAM Press.
I. Duff, Direct methods, Technical report TR/PA/98/28, July 29, 1998, CERFACS.
H.W. Berry and A. Sameh (1988), Multiprocessor schemes for solving block tridiagonal linear systems, *The International Journal of Supercomputer Applications*, 12, 37-57.

Some references:

I. S. Duff, A. M. Ersiman and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, 1986. Reprinted 1989.

I. S. Duff, R. G. Grimes and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Software*, 15, 1-14, 1989.

J.A. George and J.W.H. Liu. *Computer solution of large sparse positive definite systems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

K.A. Gallivan, R.J. Plemmons, and A.H. Sameh (1990), Parallel algorithms for dense linear algebra computations, *SIAM Review*, 32, 54-135.

J. George and J.W.H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31, 1-19, 1989.

F.-C. Lin and K.-L. Chung (1990), A cost-optimal parallel tridiagonal system solver, *Parallel Computing*, 15, 189-199.

E. Rothberg and S.C. Eisenstat. Node selection strategies for bottom-up sparse matrix ordering. *SIAM J. Matr. Anal. Appl.*, 19, 682-695, 1998.

H. van der Vorst and K. Dekker, Vectorization of linear recurrence relations, *SIAM Sci. Stat. Comp.*, 10 (1989), 27–35.