UPPSALA
UNIVERSITET

Computational Methods for Statistics with Applications

Maya Neytcheva

Department of Information Technology

Uppsala University

UPPSALA
UNIVERSITET

Sparse matrices

Plan of the lecture:

- Sparse matrices - who are those?
- Where do sparse matrices occur in Statistical applications?
- Why are sparse matrices a topic of special interest?
- Handling sparse matrices. Sparse data formats
- Solution methods for sparse matrices
  - Direct methods
    - Fill-ins and can we get rid of them?
    - Reordering strategies
    - Sparse Cholesky factorization
    - Sparse QR, SVD
  - Iterative methods (briefly)
- Examples

UPPSALA UNIVERSITET

Large matrices

What has been and is considered as large through the years

$N(t)$

1970	200
1975	1000
1980	10000
1985	100000
1990	250000
1995	500000
2000	2000000
since 2005	500000000

What is a sparse matrix?

$$A(N \times N), \quad nnz(A) = kN, \quad 2 \leq k \leq logN$$

UPPSALA UNIVERSITET

Where do sparse matrices arise?

acoustic scattering	demography	network flow
air traffic control	economics	oceanography
astrophysics	electrical eng.	petroleum eng.
biochemical	electric nets	reactor modelling
chemical eng.	climate/pollution studies	statistics
chemical kinetics	fluid flow	structural eng
circuit physics	laser optics	survey data
computer simulations	linear programming	signal processing

UPPSALA
UNIVERSITET

Sparse matrices in Statistical applications

One application: Quantitative trait loci (QTL)
Inheritance of **quantitative traits** or polygenic inheritance refers to the inheritance of a phenotypic characteristic that varies in degree and can be attributed to the interactions between two or more genes and their environment.

Though not necessarily genes themselves, quantitative trait loci (QTLs) are stretches of DNA that are closely linked to the genes that underlie the trait in question. QTLs can be molecularly identified, for example, with PCR (Polimerase Chain reaction) or AFLP (Amplified Fragment Length Polymorphism) to help map regions of the genome that contain genes involved in specifying a quantitative trait.

This can be an early step in identifying and sequencing these genes.

UPPSALA UNIVERSITET

Sparse matrices in Statistical appl.: QTL

Description: Given an IBD matrix A (Identity By Descent) - symmetric positive semidefinite.
Needed: to minimize some log-likelihood matrix L, which boils down to a **nonlinear** solution method with approximated Hessian (HI) and gradient (GL) of L as follows:

$$HI = \begin{pmatrix} y^T PAPAPy & y^T PAPPy \\ y^T PAPPy & y^T PPPy \end{pmatrix} \quad GL = - \begin{pmatrix} tr(AP) - y^T PAPy \\ tr(P) - y^T PPy \end{pmatrix}$$

where $V = \sigma_1 A + \sigma_2 I$ and
$P = V^{-1} - V^{-1}X(X^T V^{-1} X)^{-1} X^T V^{-1}$.
Task: find the blocks in GL and HI and execute the nonlinear solver for a sequence of IBD matrices A.

IBD ...

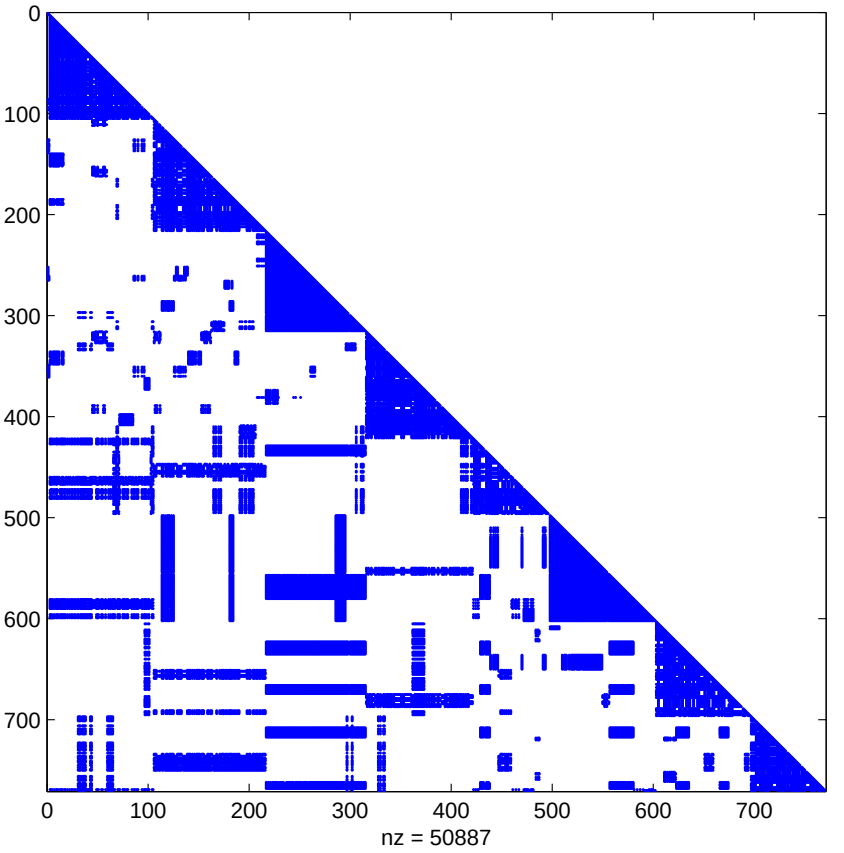
A - spsd $\Longrightarrow$ $A = A^T, eig(A) \geq 0$

$V = \sigma_1 A + \sigma_2 I \Longrightarrow$ $V = V^T, eig(V) = \sigma_1 eig(A) + \sigma_2, V^{-1}$ exists.

$P = V^{-1} - V^{-1}X(X^T V^{-1} X)^{-1} X^T V^{-1} \Longrightarrow$ $P = P^T$

UPPSALA UNIVERSITET

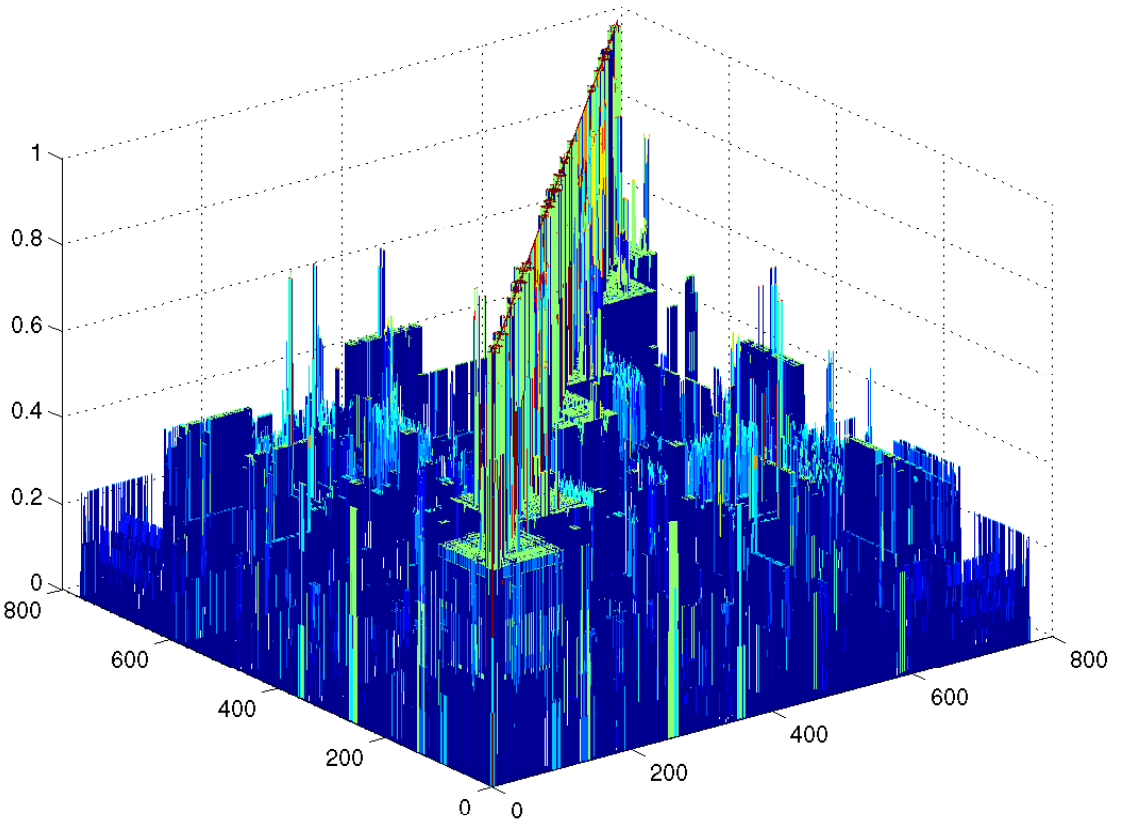
Sparse matrices in Statistical appl.: QTL



A portrait of one IBD matrix (`spy(A)`)

UPPSALA UNIVERSITET

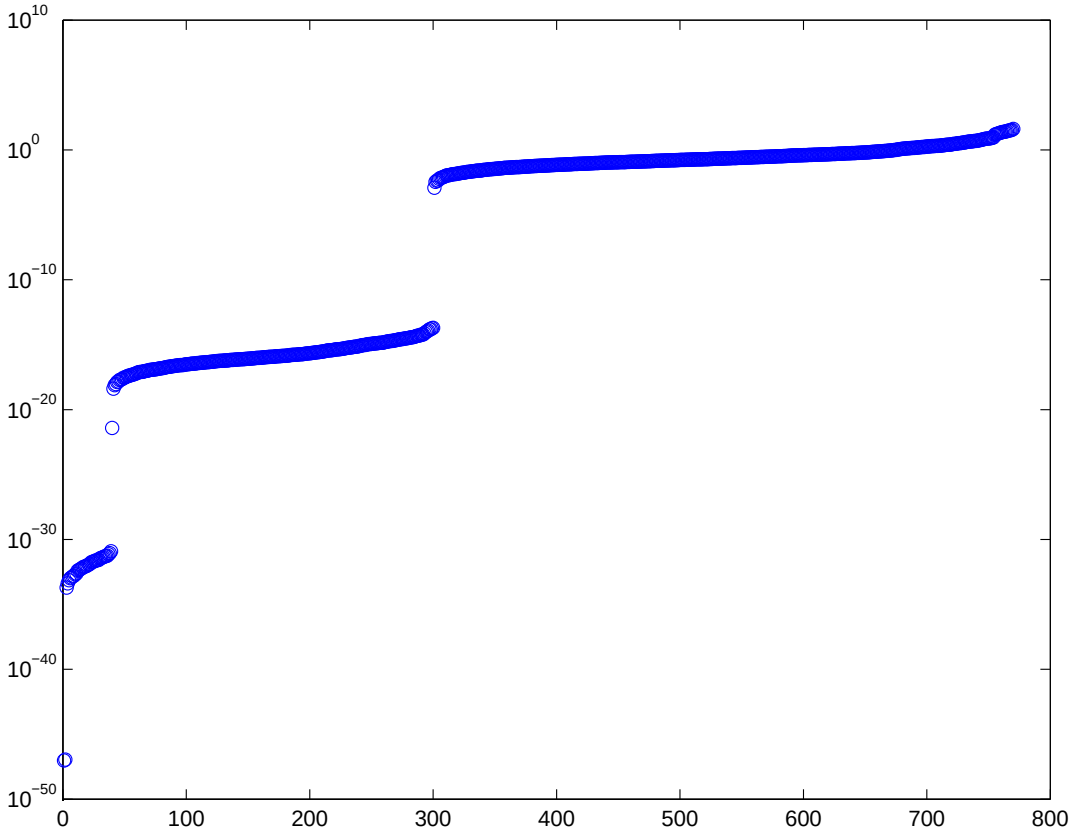
Sparse matrices in Statistical appl.: QTL



Another portrait of the same IBD matrix (`mesh(A)`)

UPPSALA UNIVERSITET

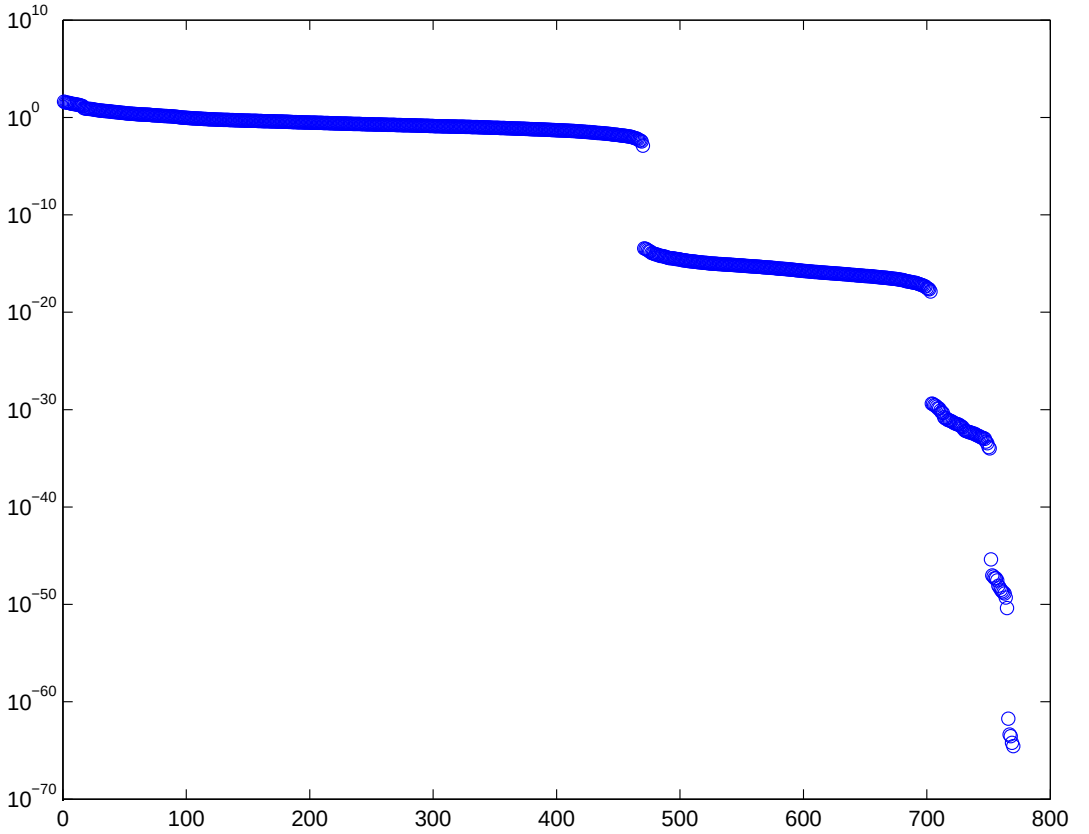
Sparse matrices in Statistical appl.: QTL



The eigenvalues of the IBD matrix (`plot(eig(A),'o')`)

UPPSALA UNIVERSITET

Sparse matrices in Statistical appl.: QTL



The singular values of the IBD matrix (`plot(svd(A),'o')`)

Major computational tasks in Statistical applications

- LS: solving the normal equation, Cholesky factorization $min\|A\mathbf{x} - \mathbf{b}\|$, $A^TA = LL^T$, $LL^T\mathbf{x} = A^T\mathbf{b}$
- LS: $A = QR$, $\mathbf{x} = R^{-1}Q_1^T\mathbf{b}$
- LS: $A = U\Sigma V^T$
- PCA (principal component analysis): SVD
- PCR (principal component regression): truncated SVD
- $A\mathbf{x} = \lambda\mathbf{x}$
- $trace(A^{-1})$

UPPSALA
UNIVERSITET

Major computational tasks in Statistical applications

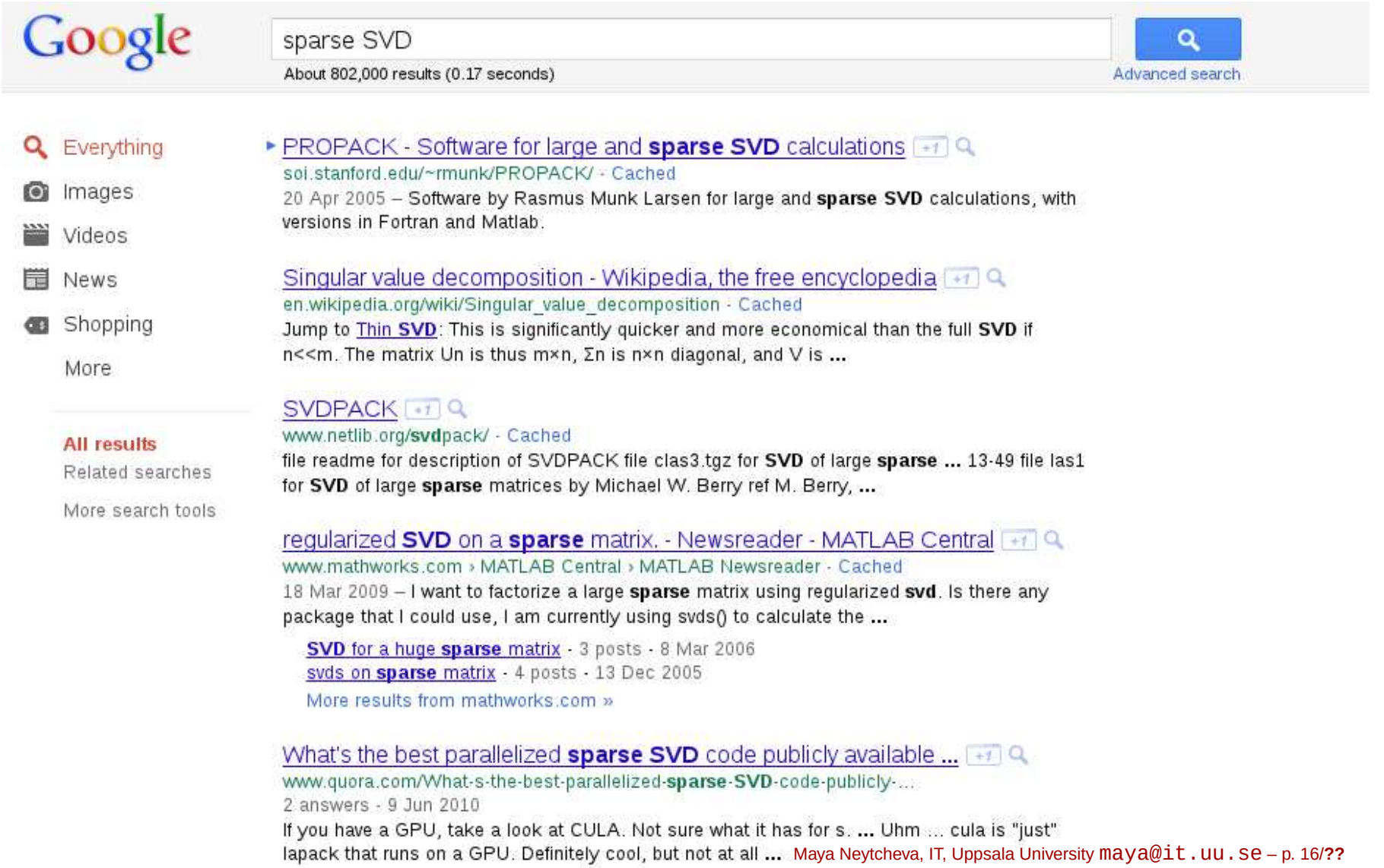
Thanks to software (Matlab, $\mathbb{R}$) and powerful computers on our desk, we do not care that much about sparse-dense etc... until we face large enough problems or we have to repeat a computational task 100, 1000, 10000 times.

The lecture today concerns storage book-keeping and programming aspects which will help to

- do the computations faster
- save computer memory

UPPSALA UNIVERSITET

Are we the only ones to be interested in that?



UPPSALA
UNIVERSITET

Before discussing sparse matrices...

we are going to look first at dense matrices...
because these are easier.

UPPSALA UNIVERSITET

Dense matrix storage schemes

Given a dense matrix $A(m, n)$.
Two main possibilities to store dense matrices:
row-wise and *column-wise*.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

First row

DRW :

a_{11}	a_{12}	$\cdots$	a_{1n}	a_{21}	a_{22}	$\cdots$	a_{2n}	$\cdots$	a_{m1}	a_{m2}	$\cdots$	a_{mn}

First column

DCW :

a_{11}	a_{21}	$\cdots$	a_{m1}	a_{12}	a_{22}	$\cdots$	a_{m2}	$\cdots$	a_{1n}	a_{2n}	$\cdots$	a_{mn}

UPPSALA
UNIVERSITET

Dense matrix storage schemes and computations

What difference does the storage scheme make wrt computations?

Dense matrix storage schemes and computations

What difference does the storage scheme make wrt computations?

Matrix-vector multiplications: $\mathbf{y} = A\mathbf{x}, A\ A(m,n)$

$$y_i = \sum_{j=1}^{n} a_{ij}x_j, \quad i = 1, \ldots, m.$$

UPPSALA UNIVERSITET

Dense matrix storage schemes and computations

What difference does the storage scheme make wrt computations?

Matrix-vector multiplications: $\mathbf{y} = A\mathbf{x}, A\ A(m,n)$

$$y_i = \sum_{j=1}^{n} a_{ij}x_j, \quad i = 1, \ldots, m.$$

A: stored row-wise (inner product scheme)

```
for i=1:m
    y(i) = 0
    for j = 1:n
        y(i) = y(i) + A(i,j) * x(j)
    end
end
```

Dense matrix storage schemes and computations

What difference does the storage scheme make wrt computations?

Matrix-vector multiplications: $\mathbf{y} = A\mathbf{x}, A\ A(m,n)$

$$y_i = \sum_{j=1}^{n} a_{ij}x_j, \quad i = 1, \ldots, m.$$

```
for i=1:m
    y(i) = 0
    y(i) = y(i) + A(i,:)  * x(:)
end
```

Dense matrix storage schemes and computations

Matrix-vector multiplications: $\mathbf{y} = A\mathbf{x}$

A: stored column-wise (outer product scheme)

```
y = 0
for j=1:n
   for i = 1:m
      y(i) = y(i) + A(i,j) * x(j)
   end
end
```

$\mathbf{y} = 0$

for $j = 1 : n$

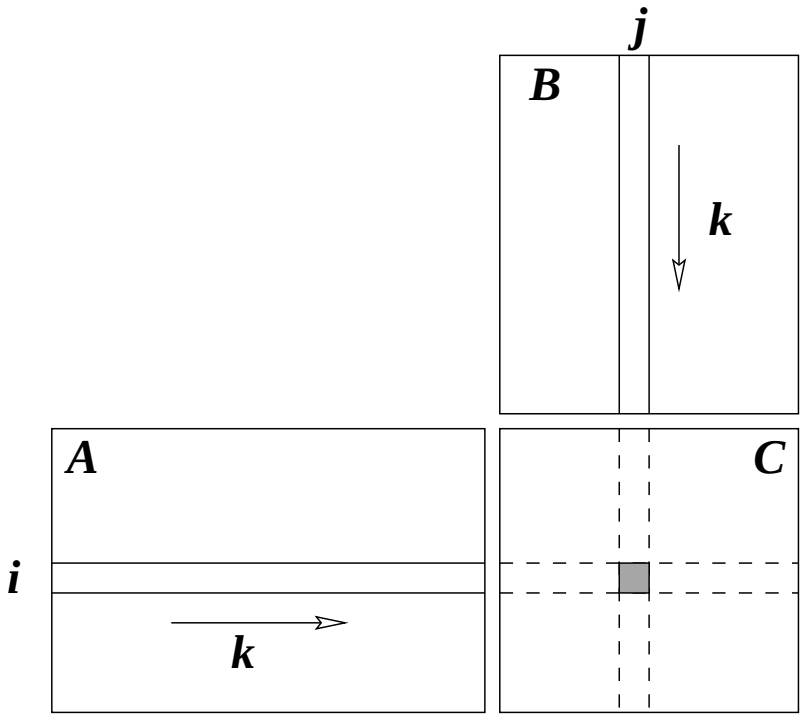
$\mathbf{y} = \mathbf{y} + x(j) * A(:, j)$

end

(vector operation)

Dense matrix-matrix multiply (ijk)

$A(m,n)*B(n,p) = C(m,p)$

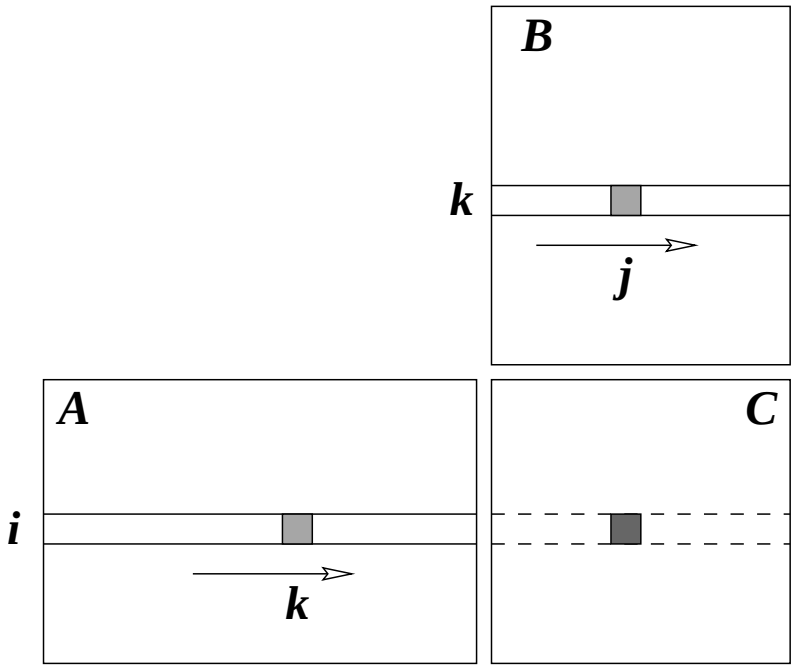



```
C = 0
for  i=1:m
   for  j=1:p
      for  k=1:n
         C(i,j) = C(i,j) + A(i,k)*
      end
   end
end
```

Scalar-product type of computation

Dense matrix-matrix multiply (ijk)

$A(m,n)*B(n,p) = C(m,p)$

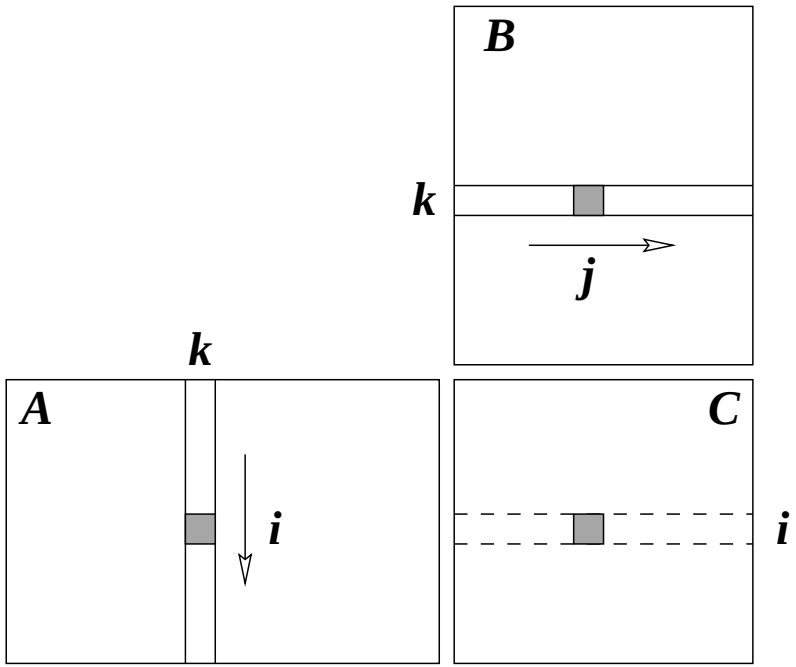



```
C = 0
for  i=1:m
    for  k=1:n
       for  j=1:p
          C(i,j) = C(i,j) + A(i,k)*B
       end
    end
end
```

Outer-product type of computation (’ikj’ - row.wise and ’jki’ - column-wise)

Dense matrix-matrix multiply (ijk)

$A(m,n)*B(n,p) = C(m,p)$




```
C = 0
for  k=1:n
   for  i=1:m
      for  j=1:p
         C(i,j) = C(i,j) + A(i,k)*B
      end
   end
end
```

accumulated update form

Dense matrix storage schemes and computations

Bottom line:

- The storage scheme of a dense matrix affects the order how the matrix entries are accessed in the computer memory.
  This may have a significant effect on the performance of an algorithm since the memory accesses are much slower than arithmetic operations
- One storage scheme is better for some operations and not so preferable for other operations (A^T).

UPPSALA
UNIVERSITET

Sparse matrix storage schemes

UPPSALA UNIVERSITET

Sparse matrix storage schemes

Coordinate scheme:

$$A = \begin{bmatrix} 0 & 1 & 2 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 6 & 0 & 0 & 0 \end{bmatrix}$$

V :

1	2	3	4	5	6

I :

1	1	2	2	3	4

J :

2	3	1	2	3	1

Advantages and disadvantages

UPPSALA UNIVERSITET

Sparse matrix storage schemes

Diagonal-wise storage scheme:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & a_{25} & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & a_{36} \\ 0 & 0 & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix} \qquad V = \begin{bmatrix} 0 & a_{11} & a_{12} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{25} \\ a_{32} & a_{33} & a_{34} & a_{36} \\ a_{43} & a_{44} & a_{45} & 0 \\ a_{54} & a_{55} & a_{56} & 0 \\ a_{65} & a_{66} & 0 & 0 \end{bmatrix}$$

OF :

-1	0	1	3

UPPSALA UNIVERSITET

Sparse matrix storage schemes

Sparse compressed schemes: $A = \begin{bmatrix} 0 & 1 & 2 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 6 & 0 & 0 & 0 \end{bmatrix}$

V :	1	2	3	4	5	6
C :	2	3	1	2	3	1
R :	1	3	5	6	7	

(a) SRW

V :	3	6	1	4	2	5
R :	2	4	1	2	3	3
C :	1	3	5	7	7	

(b) SCW

UPPSALA UNIVERSITET

Sparse matrix storage schemes

Jagged diagonal storage: The Jagged Diagonal Storage format can be useful for the implementation of iterative methods on parallel and vector processors. Like the Compressed Diagonal format, it gives a vector length essentially of the size of the matrix. It is more space-efficient than CDS at the cost of a gather/scatter operation.

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

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

Jagged diagonals, cont.

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

vals	6	9	3	10	9	5;	7	6	8	-3	13	-1;	5	-2	7	1;	4;
cols	2	2	1	1	5	5;	4	3	3	2	6	6;	5	5	4	4;	6;

perm	4	2	3	1	5	6

jd_ptr	1	7	13	17

UPPSALA
UNIVERSITET

Solution methods for linear systems of equations

- direct methods
- iterative methods

... issues, related to sparse matrices.

Direct methods:

$$A = LU,\ LU\mathbf{x} = \mathbf{b},\ L\mathbf{y} = \mathbf{b},\ U\mathbf{x} = \mathbf{y}$$

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.

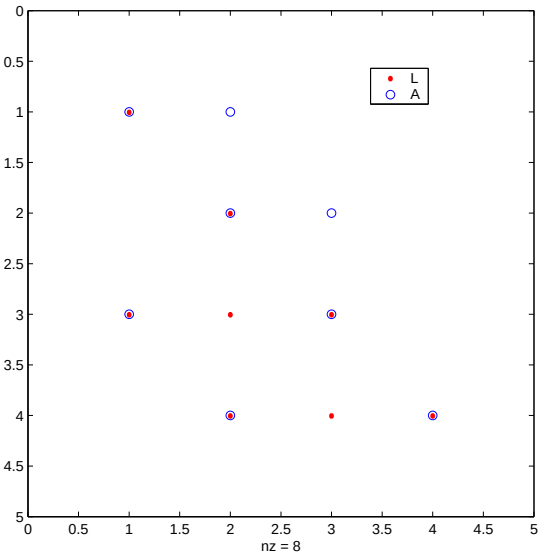
Type of matrix A	Factor	LU solve	Memory
general dense	$2/3n^3$	$O(n^2)$	$n(n+1)$
symmetric dense	$1/3n^3$	$O(n^2)$	$1/2n(n+1)$
band matrix $(2q+1)$	$O(q^2n)$	$O(qn)$	$n(2q+1)$

UPPSALA UNIVERSITET

The reason to consider especially

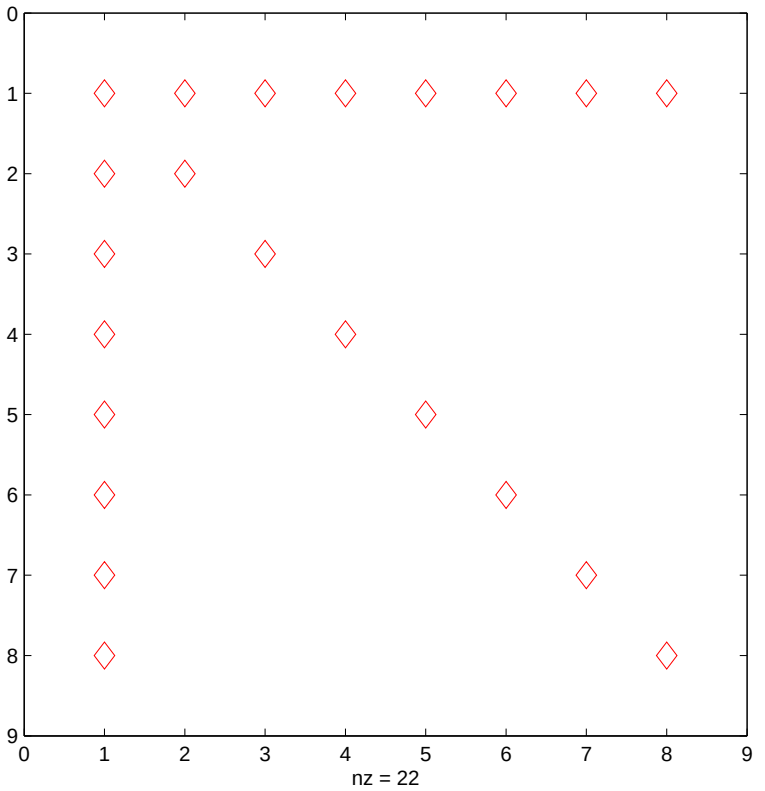
factorizations of sparse matrices is the effect of *fill-in*, namely, obtaining nonzero entries in the LU factors in positions where $A_{i,j}$ is zero. This is easy to be seen from the basic Gaussian elimination operation:

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

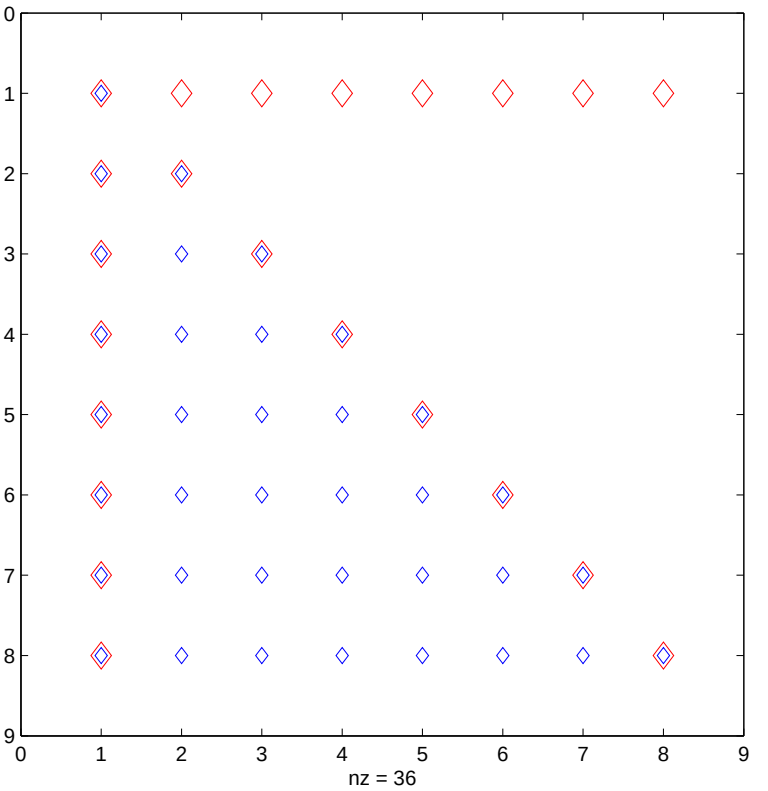


UPPSALA UNIVERSITET

Effect on sparsity structure on factorization:



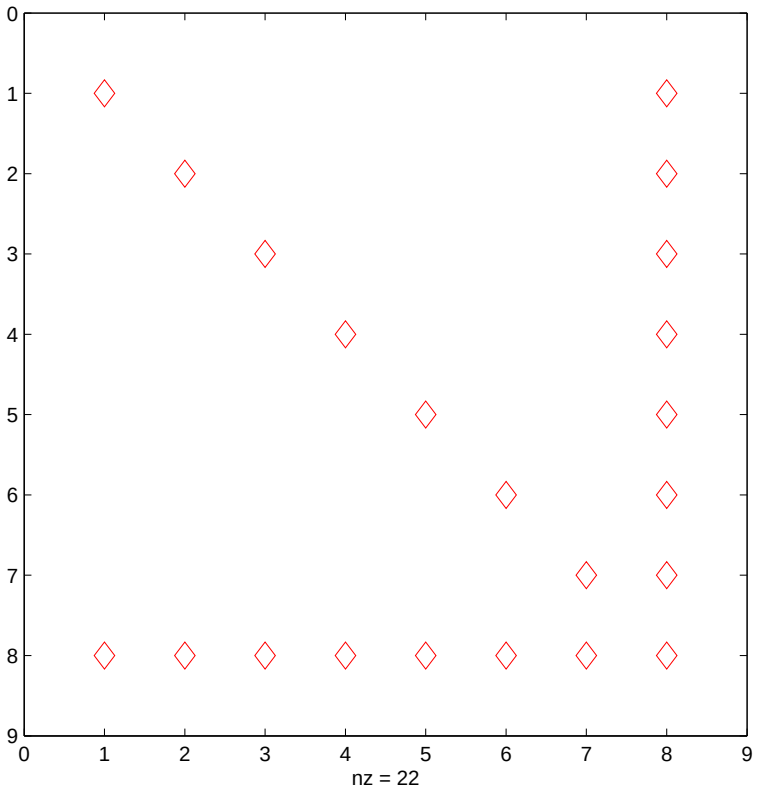
(c) Arrow matrix



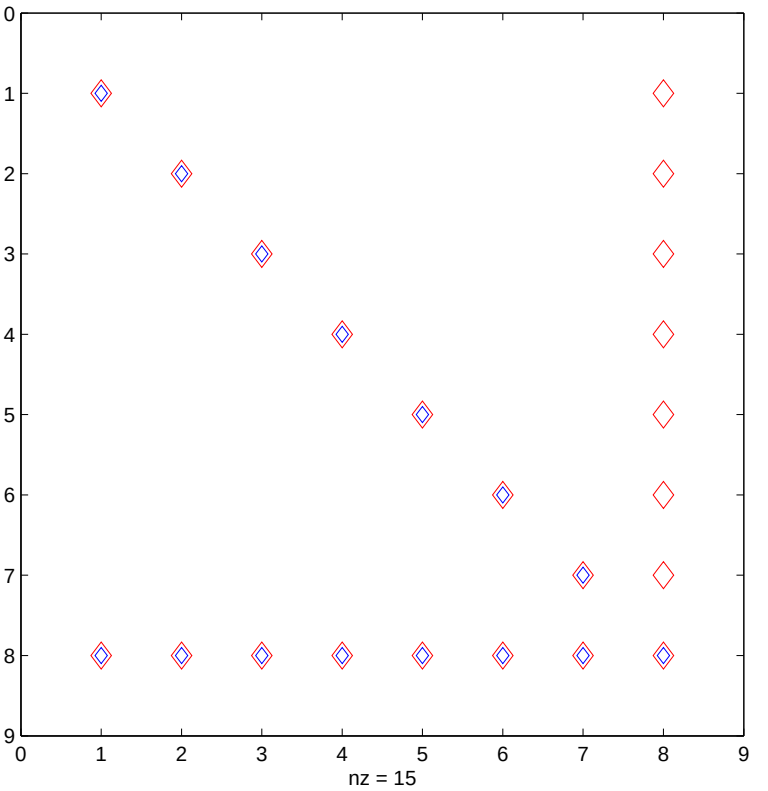
(d) The structure of the L-factor

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

Effect on sparsity structure on factorization



(e) Arrow matrix permuted



(f) The structure of the L-factor

We can permute the matrix A first and then factorize!

UPPSALA
UNIVERSITET

We pose now 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 so-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?

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 of A	$nnz(A)$	Time in sec	
		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)

UPPSALA
UNIVERSITET

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;

UPPSALA
UNIVERSITET

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.

UPPSALA
UNIVERSITET

```
n=500;
R=sprand(n,n,0.25);I=speye(n);b=rand(n,1);A=I+R;AF=full(A);
tic,x=A\b;toc
Elapsed time is 0.072157 seconds.
tic,x=AF\b;toc
Elapsed time is 0.159156 seconds.

n=5000;
tic,x=A\b;toc
Elapsed time is 10.729183  seconds.
tic,x=AF\b;toc
Elapsed time is  6.378155 seconds.

n=10000;
tic,x=A\b;toc
Elapsed time is 308.037052 seconds.
tic,x=AF\b;toc
Elapsed time is 65.238327 seconds.
```

UPPSALA
UNIVERSITET

Two possible situations can 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*).

UPPSALA
UNIVERSITET

"Given-the-matrix" strategy

UPPSALA
UNIVERSITET

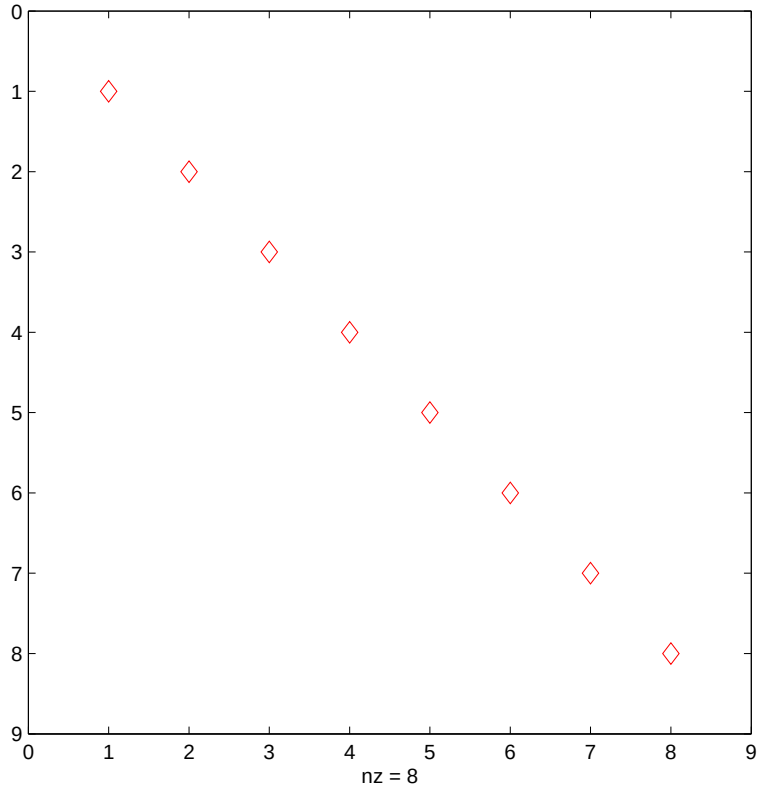
Given-the-matrix strategy

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?

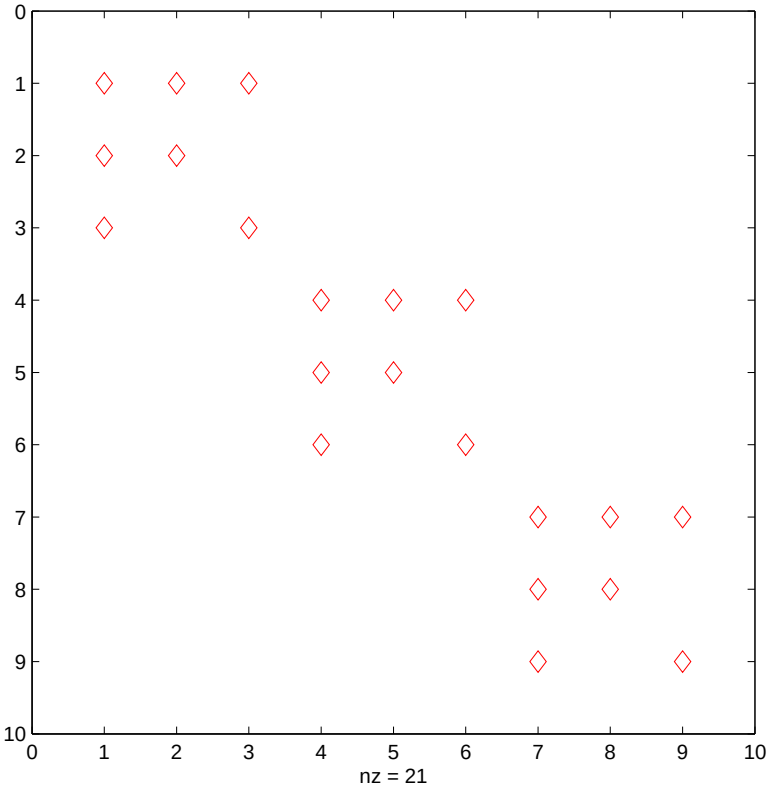
UPPSALA UNIVERSITET

Given-the-matrix strategy

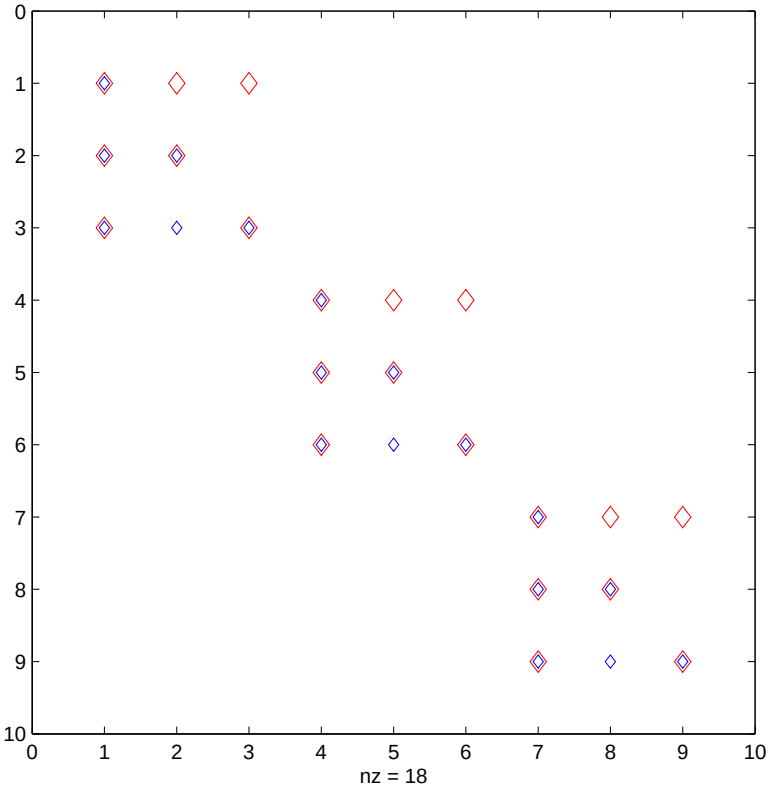


(g) Diagonal matrix

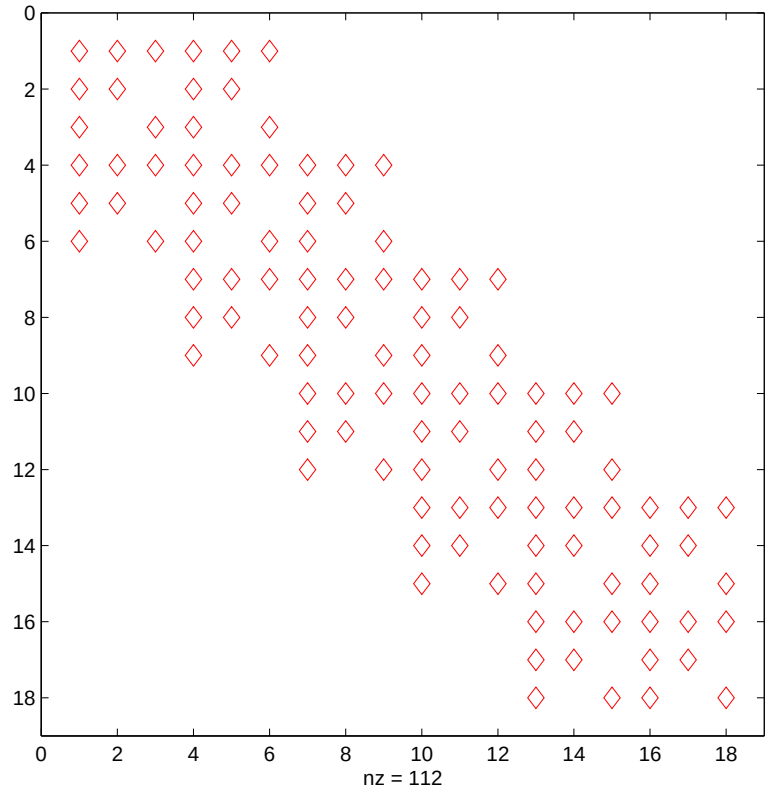
- diagonal
- block-diagonal
- block-tridiagonal
- arrow matrix
- band matrix
- block-triangular



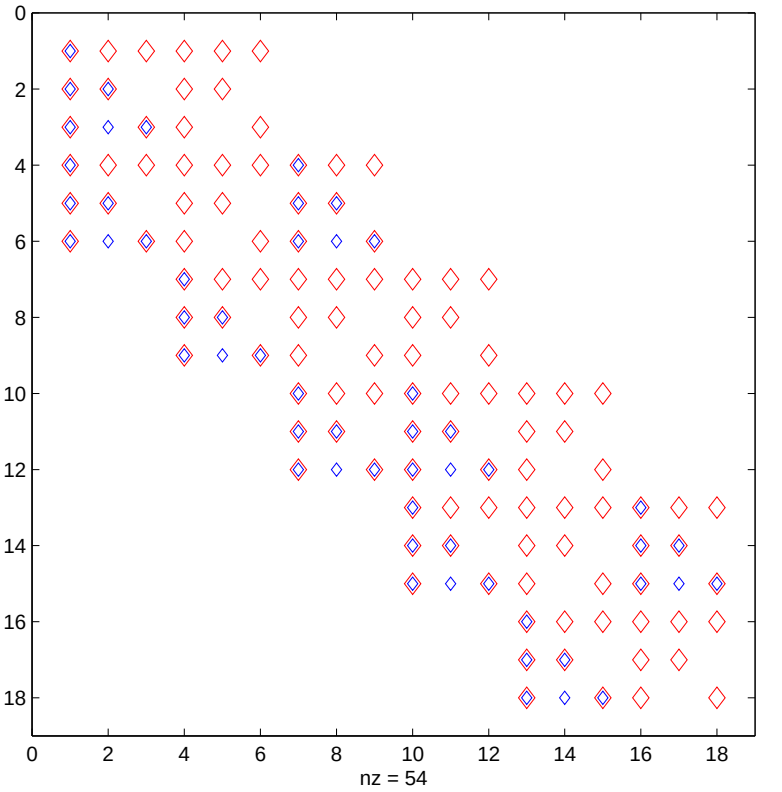
(h) block-diagonal matrix



(i) The structure of the L-factor



(j) Block-tridiagonal matrix



(k) The structure of the L-factor

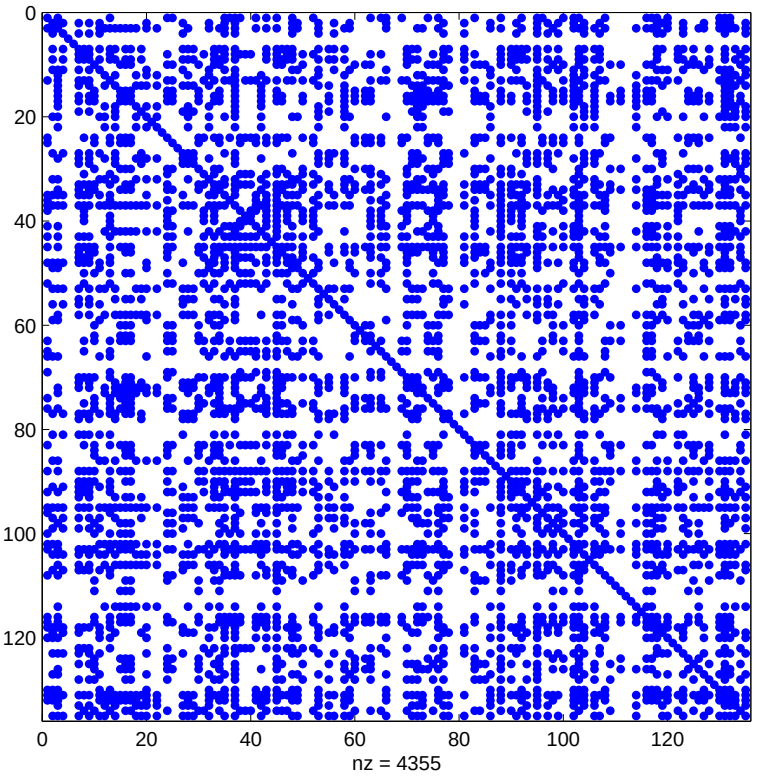
UPPSALA
UNIVERSITET

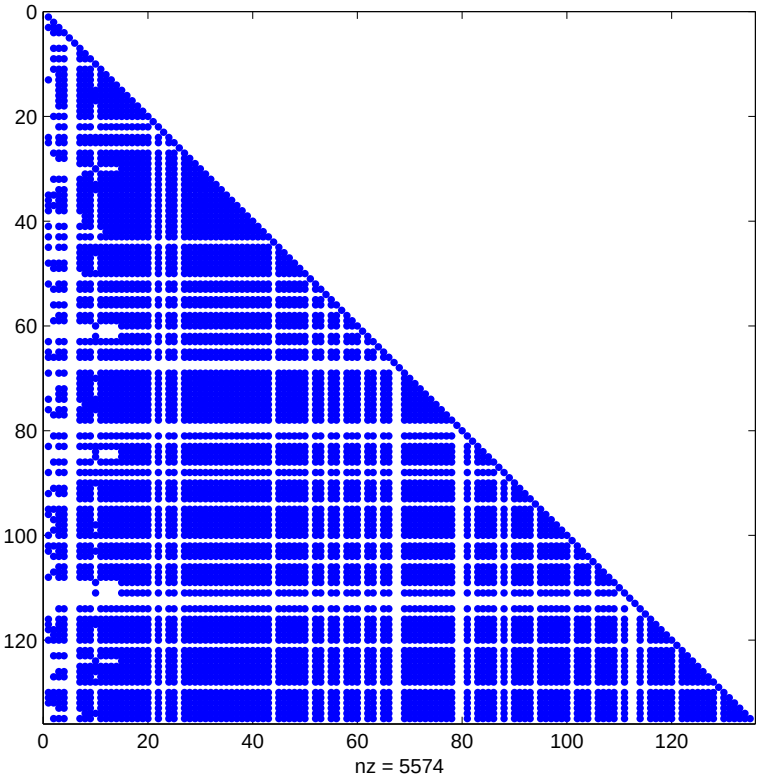
We will consider the case of symmetric matrices ($P = Q$) and three popular methods based on manipulations on the graph representation of the matrix.

- (generalized) reverse Cuthill-McKee algorithm (1969);
- nested dissection method (1973);
- minimum degree ordering (George and Liu, 1981) and variants.

UPPSALA UNIVERSITET

A matrix from somewhere





UPPSALA
UNIVERSITET

Generalized Reverse Cuthill-McKee (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 $w_i = v_{n+1-i}$.

UPPSALA
UNIVERSITET

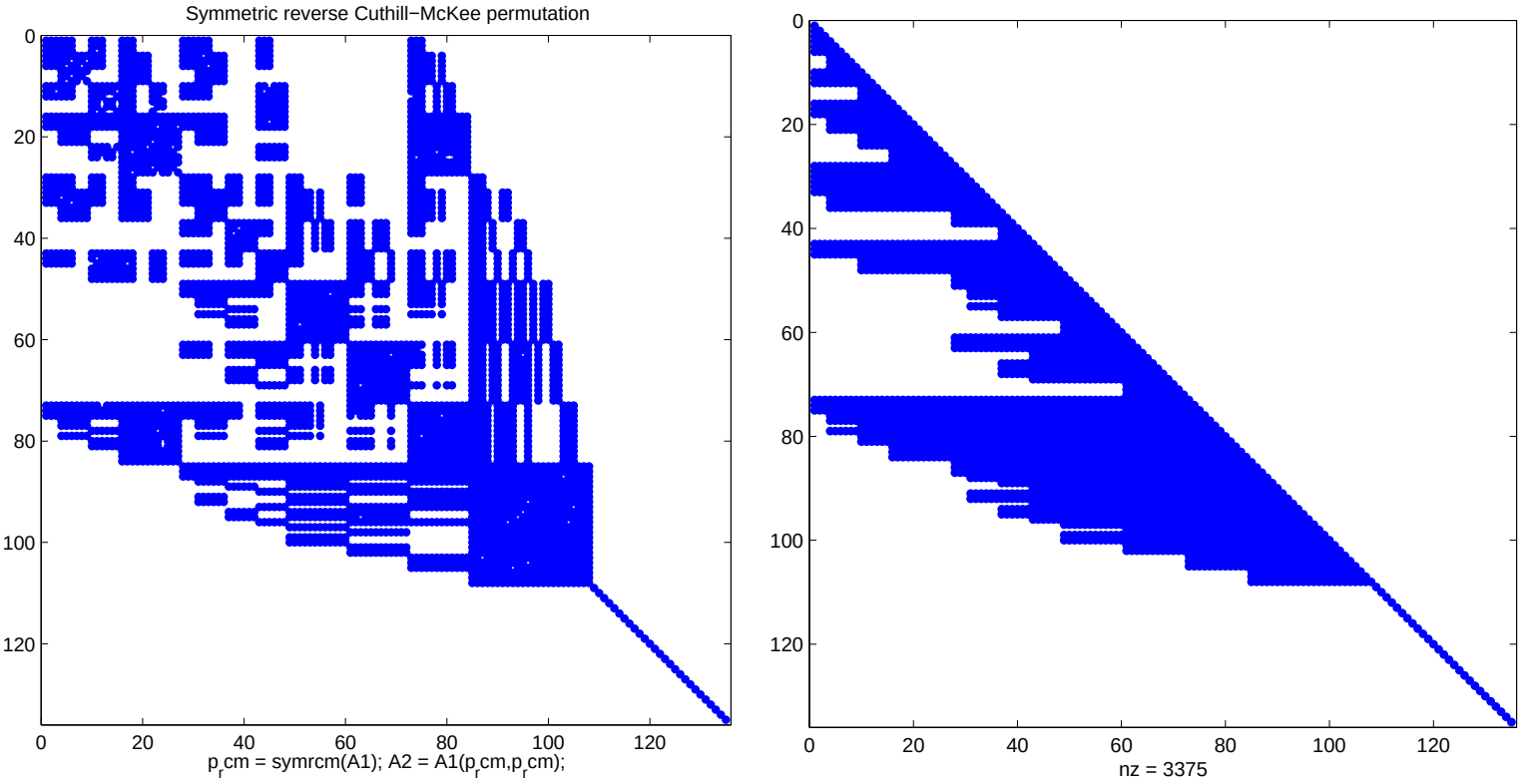
Generalized Reverse Cuthill-McKee (RCM)

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 (RCM)



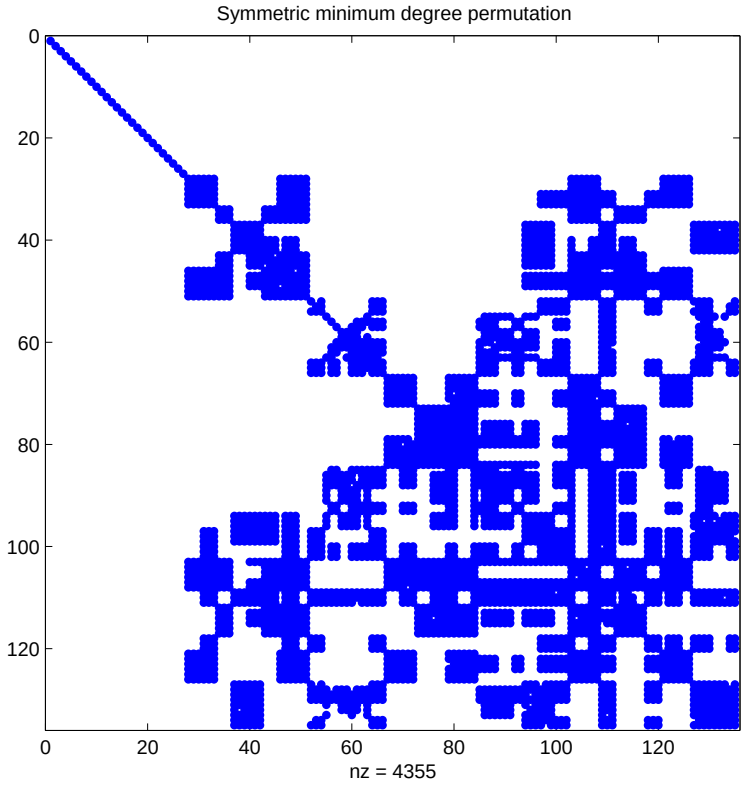
UPPSALA
UNIVERSITET

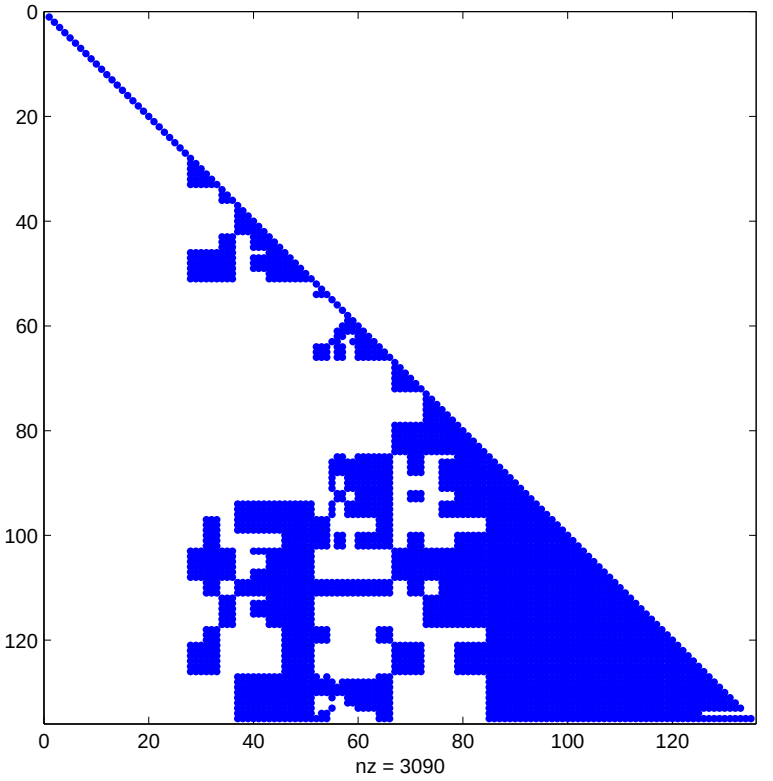
The Quotient Minimum Degree (QMD)

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.

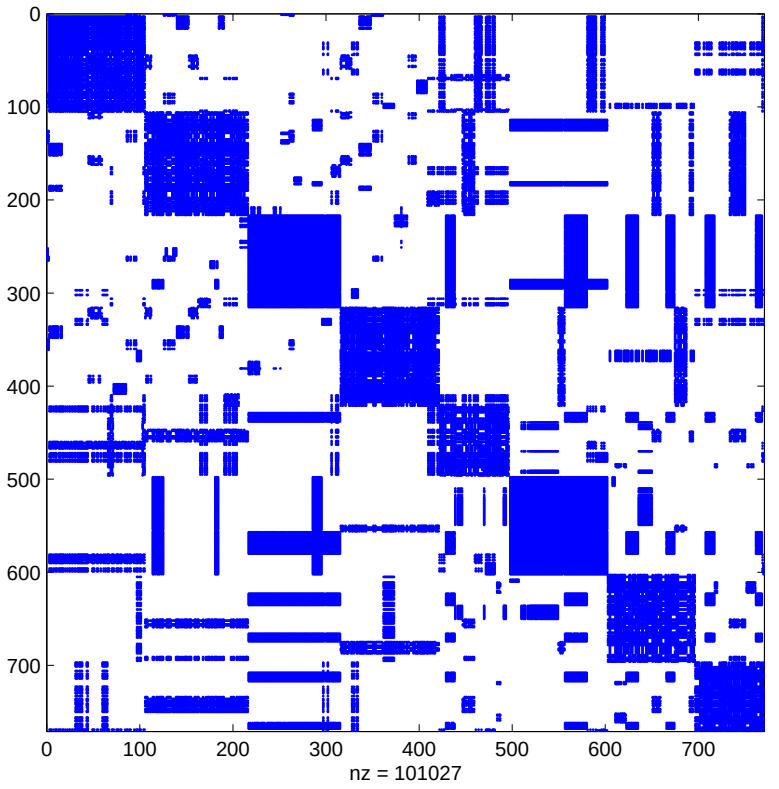
The Quotient Minimum Degree (QMD)

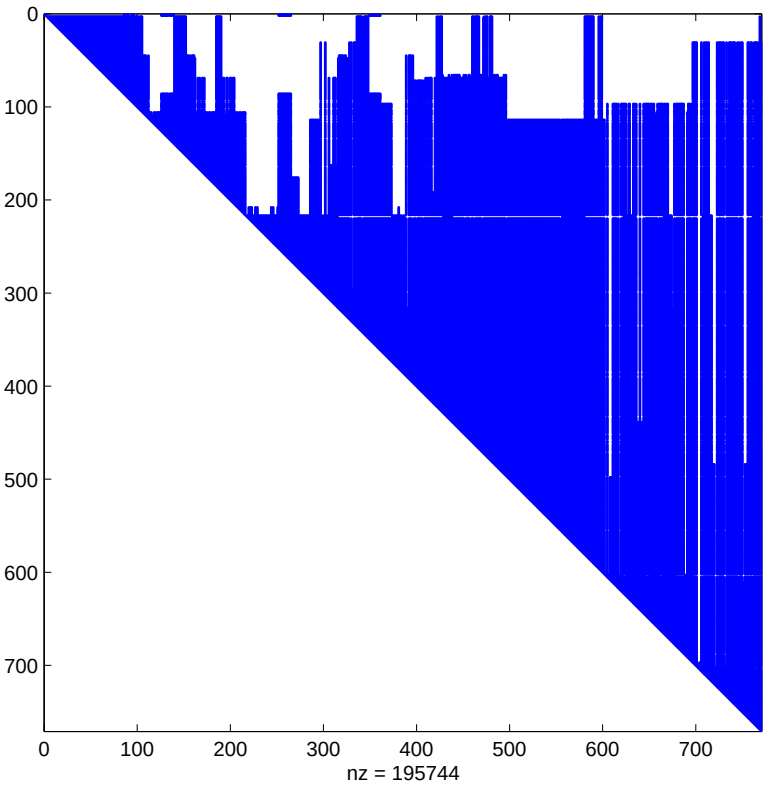




UPPSALA UNIVERSITET

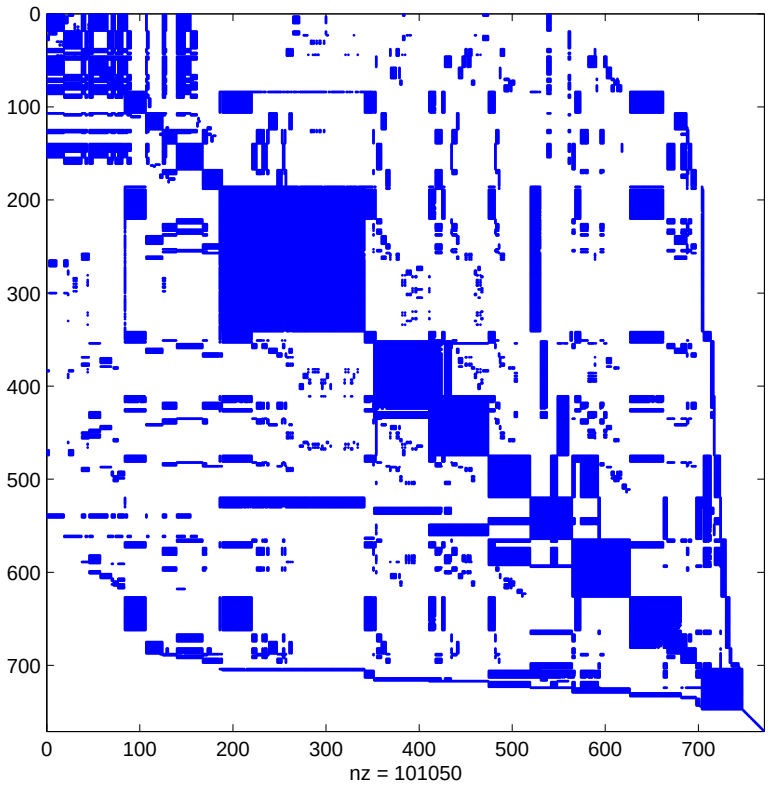
IBD matrix: X20

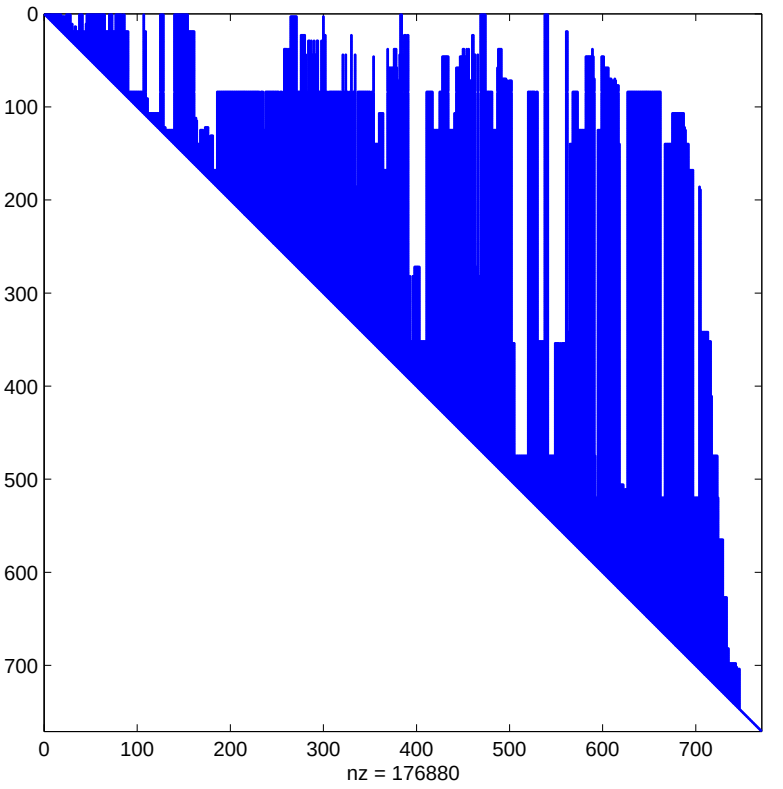




UPPSALA UNIVERSITET

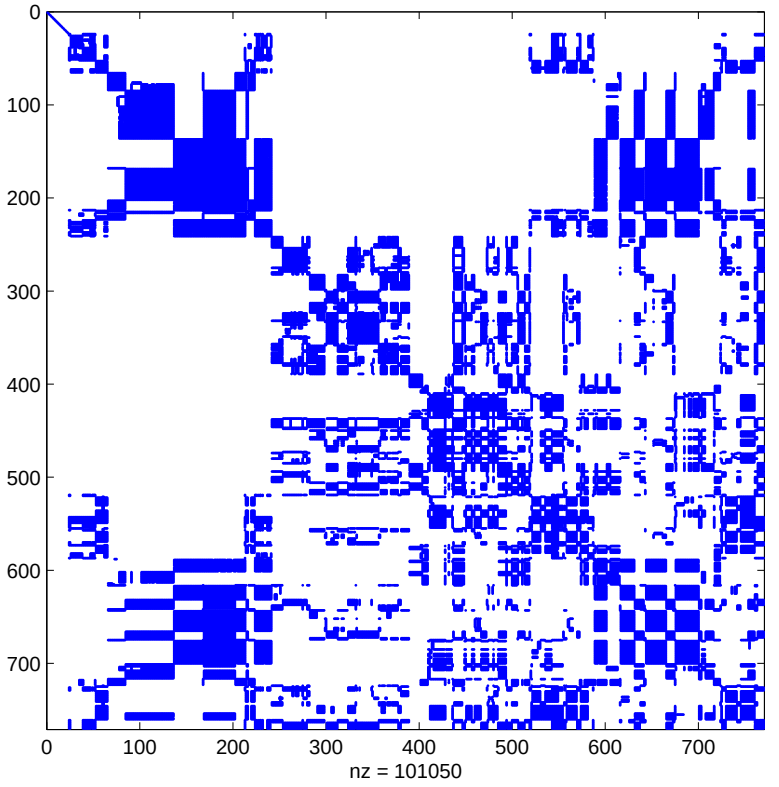
IBD matrix: X20: RCM

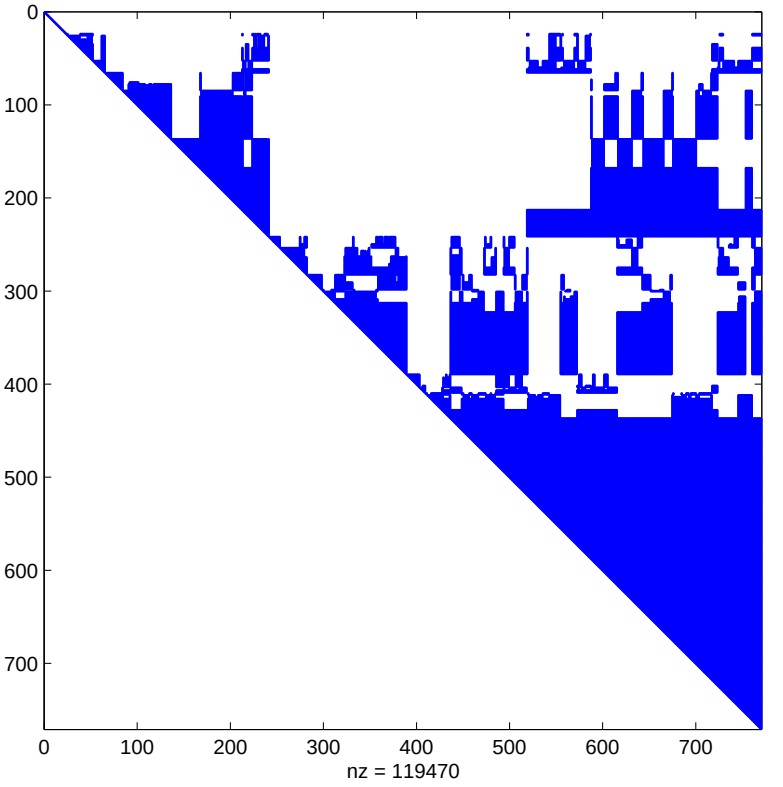




UPPSALA UNIVERSITET

IBD matrix: X20: MMD





UPPSALA UNIVERSITET

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 $(1,2)$

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

UPPSALA
UNIVERSITET

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 some class of model 2D problems originating from discretized PDEs. The Cholesky factor contains $O(m^2 log_2 m)$ nonzero entries. This is the *best low order bounds* derived for direct elimination methods.

UPPSALA
UNIVERSITET

A comparison...

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

Comparison results, problem size $n = 92862$

UPPSALA UNIVERSITET

Direct methods, 2D problem:

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
$n = 92862$ $nzA = 1254552$					
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

UPPSALA UNIVERSITET

Direct methods, a second 2D problem

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
$n = 210562$ $nzA = 2917310$					
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

Method	Ordering	Factorization	Solution	Total time	True res.
$n = 675 \quad nzA = 32817$					
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

UPPSALA
UNIVERSITET

"Given-the-problem" strategy

UPPSALA
UNIVERSITET

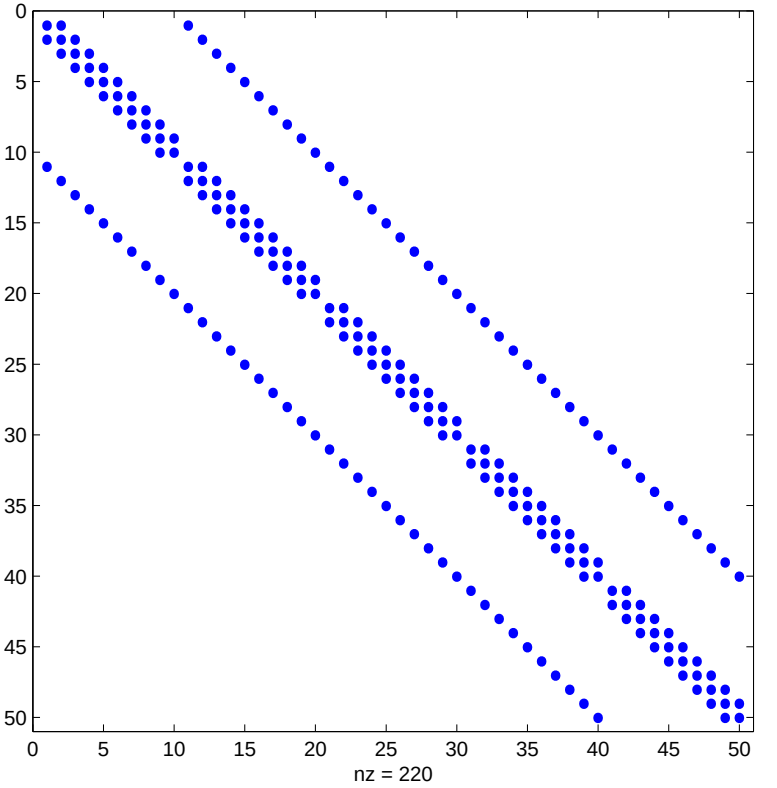
Given-the-problem strategy

Assume we know the origin of the linear system of equations to be solved.
Example: the problem originates from a numerically discretized (system of) PDEs, and we know the domain of definition of the problem (Ω), its geometrical 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.

UPPSALA UNIVERSITET

41	42	43	44	45	46	47	48	49	50
31	32	33	34	35	36	37	38	39	40
21	22	23	24	25	26	27	28	29	30
11	12	13	14	15	16	17	18	19	20
1	2	3	4	5	6	7	8	9	10

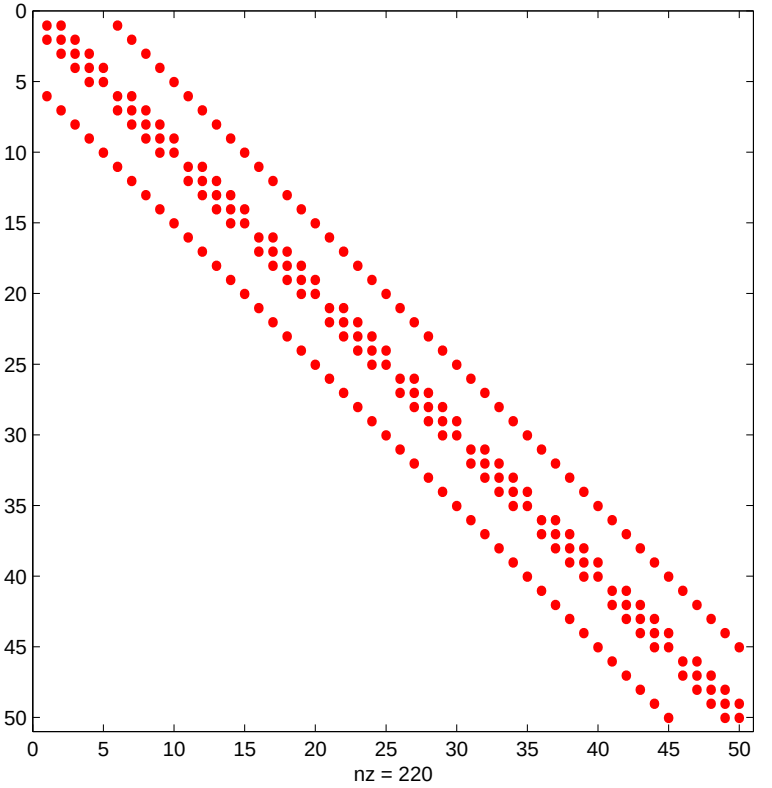
(l) Row-wise ordering



(m) The structure of the matrix A

5	10	15	20	25	30	35	40	45	50
4	9	14	19	24	29	34	39	44	49
3	8	13	18	23	28	33	38	43	48
2	7	12	17	22	27	32	37	42	47
1	6	11	16	21	26	31	36	41	46

(n) Column-wise ordering



(o) The structure of the matrix A

UPPSALA
UNIVERSITET

Major computational tasks in Statistical applications

- Cholesky factorization $\surd$
  $min\|A\mathbf{x} - \mathbf{b}\|$, $A^TA = LL^T$, $LL^T\mathbf{x} = A^T\mathbf{b}$
- LS: $A = QR$, $\mathbf{x} = R^{-1}Q_1^T\mathbf{b}$
- LS: $A = U\Sigma V^T$
- PCA (principal component analysis): SVD
- $A\mathbf{x} = \lambda\mathbf{x}$
- $trace(A^{-1})$

UPPSALA
UNIVERSITET

Sparse QR

$$A = LL^T$$

$$A = QR, \text{ then } A^TA = R^TQ^TQR = R^TR!$$

Thus, if we know R, we have factorized A^TA!
But: if A is sparse, we want that R is as sparse as possible.
This is achieved by performing symbolic factorization of A^TA.

Sparse SVD, Matlab

The Matlab SVD implementation follows that of LINPACK, which is for general dense matrices.
To find some of the singular values (largest or smallest) of a large sparse matrix, one can use svds.
svds(A,k) uses eigs to find the k largest magnitude eigenvalues and corresponding eigenvectors of

$$B = \begin{bmatrix} 0A \\ A^T 0 \end{bmatrix}$$

.

Sparse SVD, Matlab

Demo:

```
load 20.dat
A=X20+X20';
for k=1:770, A(k,k)=A(k,k)/2;end
S=svd(A);
S=svd(full(A));
S=svds(A);
```

```
S=svds(A,5,0);
S=svds(A,5,0.01);
S=svds(A,5,0.001);
S=svds(A,5,0.0001);
S=svds(A,5,0.000001);
```

UPPSALA
UNIVERSITET

Iterative solution methods

UPPSALA
UNIVERSITET

Sparse matrices in Statistical applications

- Genetic mapping of quantitative traits
- Sparse inverse of covariance matrix for QTL effects with incomplete marker data
  R. Mark Thallman, Kathryn Hanford, Stephen Kachman, L. Dale Van Vleck
  Statistical Applications in Genetics and Molecular Biology, 2005
- Sparse spatial autoregression
- Pace and Barry (1997), "Kriging with Large Data Sets," Communications in Statistics, Simulation and Compassion
  The authors discuss sparse krigging in Communication in Statistics, Simulation and Computation. Using published estimates on a spherical variogram they solve the estimates 432 times as fast as using more conventional solution techniques.

UPPSALA
UNIVERSITET

Sparse matrices in Statistical applications

- Barry and Pace (1999), "Monte Carlo Estimates of the Log Determinant of Large Sparse Matrices," Linear Algebra and its Applications
  The authors devise a means of estimating the log-determinant of large, sparse matrices. Estimation of the log-determinant of the variance-covariance matrix (or its inverse) allows maximum likelihood estimation of large-scale spatial statistical problems. Most importantly, the article shows a way of providing confidence intervals for the estimate and show these work via a coverage study. To illustrate the potential of the estimator, we estimated the log-determinant of a 1,000,000 by 1,000,000 matrix (on a Pentium 133 MHz machine). The estimator has a simple form and its performance depends only upon the degree of sparsity and not its pattern. Source code and executable code for it resides in SpaceStatPack.

UPPSALA
UNIVERSITET

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.

UPPSALA
UNIVERSITET

ℝ - Matlab data exchange

Some practical issues

ℝ - Matlab: sparse matrix exchange

```
> M=matrix(1:12,3,4)
> M
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
write.matrix(M,file="M.dat",sep=" ")
```

```
>> load M.dat
>> M
     1     4     7    10
     2     5     8    11
     3     6     9    12
```

Matlab-ℝ : sparse matrix exchange

Source file:

```
>> N=M/2
N =
    0.5000    2.0000    3.5000    5.0000
    1.0000    2.5000    4.0000    5.5000
    1.5000    3.0000    4.5000    6.0000
>> save N.dat -ASCII N
```

```
> N0=read.csv("N.dat",sep="",header=F)
> is.matrix(N0)
[1] FALSE
> N=as.matrix(N0)
> N
     V1  V2  V3  V4
[1,] 0.5 2.0 3.5 5.0
[2,] 1.0 2.5 4.0 5.5
[3,] 1.5 3.0 4.5 6.0
```

Matlab-ℝ : sparse matrix exchange

Source file: `20.dat`
Read in `Matlab`:

```
>> load 20.dat
>> size(X20)
ans =   770   770

>> X20sp=sparse(X20);
>> whos
  Name          Size                 Bytes  Class     Attributes
  X20         770x770             4743200  double
  X20sp       770x770              613728  double    sparse
```

Matlab-ℝ : sparse matrix exchange

Source file: `20.dat`
Read in ℝ :

```
> library("MASS")
> X20=read.delim("20.dat")
> dim(X20)
[1] 769   1
> AL=read.csv("20.dat",sep="",header=F)
> dim(AL)
[1] 770 770
> is.matrix(AL)
[1] FALSE
> AL=as.matrix(AL)
> AU=t(AL)
> A=AL+AU
> S=svd(A)
> plot(S$d)
```

Matlab-ℝ : sparse matrix exchange

Export the triangular factor from ℝ to Matlab:

```
> is.numeric(U)
[1] TRUE
> write.table(U,file = "Fname", quote = FALSE, sep = "")
```

Matlab-ℝ : sparse matrix exchange

Experience with ℝ - spam

```
library(spam)
> N0=read.csv("A20.dat",sep="",header=F)
> is.matrix(N0)
[1] FALSE
> A=as.matrix(N0)
> dim(A)
[1] 770 770
> Asp=as.spam(A)
> is.spam(Asp)
[1] TRUE
> D=diag.spam(x=10, 770, 770)
> B=Asp+D
> system.time(U=chol(B))
> b <- rnorm(770)
> q=solve(B,b)
> Q=chol(B)
> spam.options( imagesize=1000)
> display(B)
```

UPPSALA
UNIVERSITET

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.