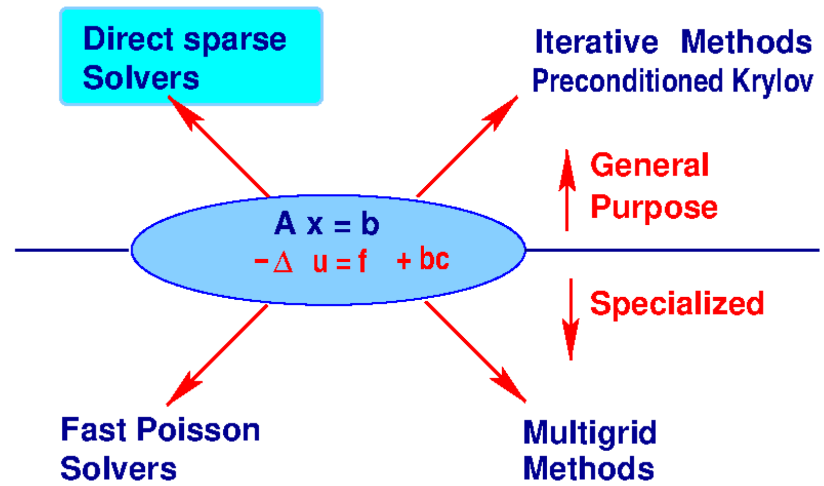
## *SPARSE GAUSSIAN ELIMINATION*

*(introduction)*

- *Gaussian elimination - dense case*
- *Variants*
- *Main computational kernels*
- *Gaussian elimination in sparse case: a preview*
- *Graph model of elimination*

## *Solving sparse systems today*






*Background.* Three types of methods:

➤ Direct methods : based on sparse Gaussian eimination, sparse Cholesky,..

➤ Iterative methods: compute a sequence of iterates which converge to the solution - preconditioned Krylov methods..

➤ Special purpose methods: Multigrid, Fast-Poisson solvers, ...

**Remark:** The first 2 classes of methods have always been in competition.

➤ 40 years ago solving a system with $n = 10,000$ was a challenge [Now you can solve this in a fraction of a second on a laptop]



Quotation from R. Varga's book on iterative methods [1962]

> *"As an example of the magnitude of problems that have been successfully solved by cyclic iterative methods, the Bettis Atomic Power Laboratory of the Westinghouse Electric Corporation had in daily use in 1960 a 2-dimensional program which would treat as a special case Laplacean-type matrix equations of order 20,000."*

He adds in footnote: (paraphrase) the program was written for the Philco-2000 computer which had 32,000 words of core storage (32Kwords!). "Even more staggering": Bettis had a 3-D code which could treat coupled matrix equations of order 108,000.

➤ Sparse direct methods made huge gains in efficiency. They are very competitive for 2-D problems.

➤ 3-D problems lead to more challenging systems [inherent to the underlying graph]

Difficulty:

- No robust 'black-box' iterative solvers.

**At issue** Robustness in conflict with efficiency.

➤ Iterative methods are starting to use some of the tools of direct solvers to gain 'robustness'



## *Gaussian Elimination. Variants*

Recall: Gaussian Elimination has 3 nested loops.

➤ The three loop indices are denoted by k, i, j

➤ We can order each of the loops differently.

➤ A total of 6 different algorithms [more if we add blocking]: kij, kji, ijk, ikj, jki, jik.

➤ IMPORTANT: these algorithms are equivalent. Same operations are done in a different order

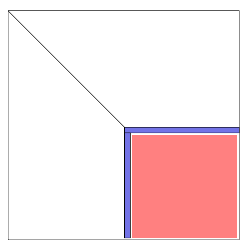
➤ Therefore same operation counts

✍1 Find the analogous algorithms for Matrix-matrix multiplication



For $k = 1 : n-1$ Do:
    For $i = k+1 : n$ Do:
        $a_{ik} := a_{ik}/a_{kk}$
        For $j := k+1 : n$ Do
            $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
        End
    End
End

**KIJ**



➤ This is the $\boxed{\text{KIJ}}$ variant of GE [hint: read the loop indices]

➤ Also known as the "Outer product" form

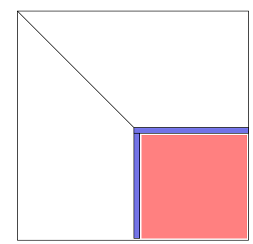
➤ Main drawback: rank-one update matrix at each step.

➤ Can also have a KJI version [flip loops $i$ and $j$]



For $k = 1 : n-1$ Do:
    For $i := k+1 : n$ Do
        $a_{ik} := a_{ik}/a_{kk}$
    EndDo
    For $j = k+1 : n$ Do:
        For $i := k+1 : n$ Do
            $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
        End
    End
End

**KJI**



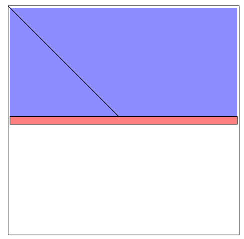
➤ Can we swap loops $k$ and $j$ ['delay the k-loop']

➤ Consider first the KIJ (row) version [flip loops $k$ and $i$]

## *Gaussian Elimination. IKJ variant*

For $i = 2, \ldots, n$ Do:
  For $k = 1, \ldots, i-1$ Do:
    $a_{ik} := a_{ik}/a_{kk}$
    For $j = k+1, \ldots, n$ Do:
      $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
    EndDo
  EndDo
EndDo

**IKJ**



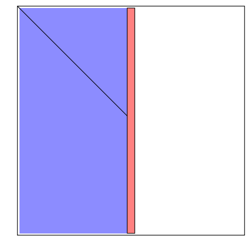
- Also known as the 'up-looking LU'
- Row-oriented 'delayed update' algorithm
- Can also have a column version [very common]



## *Gaussian Elimination. JKI or 'Left-looking LU'*

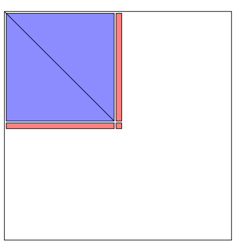
```
for j=1:n
   for k=1:j-1
      A(k+1:n,j) = A(k+1:n,j) - ...
                   A(k,j)*A(k+1:n,k);
   end
   // i loop:
   A(j+1:n,j) = A(j+1:n,j) / A(j,j);
end
```

**JKI**

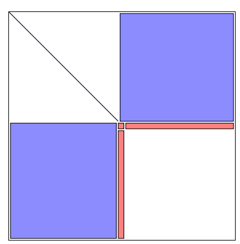




## *Other variants*



IJK variant (dot product)



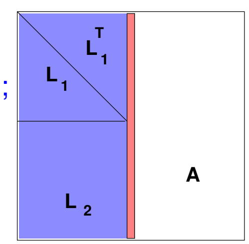
Crout

- Also: corresponding variants of Cholesky – We will only look at the adaptation of the JKI variant of Cholesky..



## *Column Cholesky (compare with JKI – Gauss)*

```
for j=1:n
   for k=1:j-1
      A(j:n,j)=A(j:n,j)-A(j:n,k)*A(j,k);
   end
   A(j,j) = sqrt(A(j,j));
   A(j+1:n,j) = A(j+1:n,j)/A(j,j);
end
```




- Column $j$ of $A$ [L-part] becomes column $j$ of $L$.

➤ We will often consider Sparse Cholesky because: 1) the SPD case is important; 2) certain aspects are simpler than Gauss; 3) Generalizations are easy..

➤ Sparse Column Cholesky: same as above algorithm but implemented in sparse mode

**Number of operations.** The total number of multiplications required to compute the Cholesky factor $L$ of a matrix $A$ is given by

$$\sum_{k=1}^{n-1} (\eta_k^2 - 1)$$

where $\eta_k$ is the number of nonzero entries in the $k$-th column of $L$



**Proof:**

➤ Consider only the KIJ version of Cholesky which is equivalent.

➤ Rank-1 update at each step is $A^{(k)} = A^{(k-1)} - v_k v_k^T$, where [using matlab notation]

$$v_k = [zeros(1,k), A^{(k-1)}(k, k+1:n)]^T$$

➤ Only lower part is done - so cost is $(\eta_k - 1)\eta_k$.

➤ Must add $\eta_k - 1$ scaling operations (mult. by inverse). Total: $\eta_k^2 - 1$ ■

➤ OK – but $\eta_k$'s not known in advance. Dense case $\eta_k = n - k + 1$

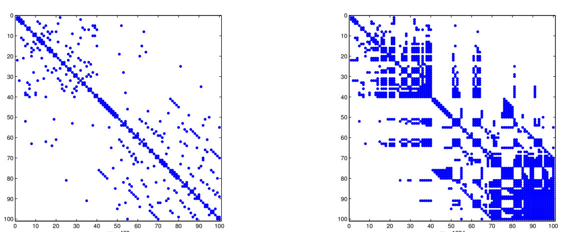
➤ Storage: $\sum_{k=1}^{n} \eta_k$



## Sparse solvers

➤ Need to develop sparse versions: sparse operations

➤ Major new consideration: Fill-in.




➤ These variants will lead to different computation kernels



## Gaussian Elimination. Graph model

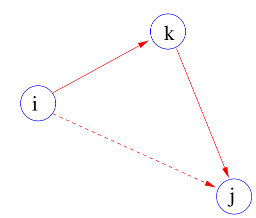
➤ What happens to the graph when a node is eliminated?

Eliminating a node $x$ in Gaussian elimination amounts to
1. Deleting node $x$ and its adjacent edges from the graph
2. Adding edges between any two nodes that were adjacent to $x$.

➤ Notation: $A^{(k)}$ = matrix at $k$-th step of elimination

*Another viewpoint:* At step $k$ of Gaussian Elimination, a fill-in is introduced in position $(i, j)$ iff

$$a_{ik}^{(k-1)} \neq 0 \;\&\; a_{kj}^{(k-1)} \neq 0$$



**A seminal paper:**

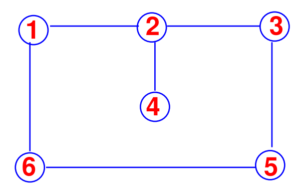
S. Parter, "*The use of linear graphs in Gauss elimination*", SIAM review, vol. 3, (1961), pp. 119-130.

➤ Gave a major impetus to the use of the graph theory approach to sparse matrix techniques.

➤ Foundation for some of the major ideas (e.g. elimination trees) to come even much later.

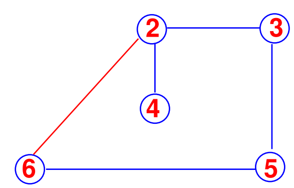


***Example:*** Gaussian elimination steps for following graph




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

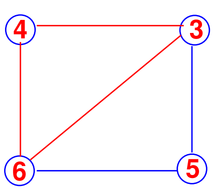
Eliminate 1:



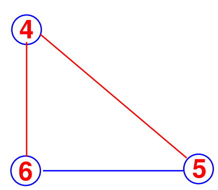

$$\begin{bmatrix} \star & & & & \\ \star & \star & & & \\ \star & & \star & & \\ & \star & & \star & \\ \bullet & & & \star & \star \end{bmatrix}$$



Eliminate 2 and then 3:



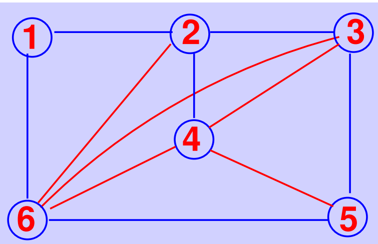

$$\begin{bmatrix} \star & & & \\ \bullet & \star & & \\ \star & & \star & \\ \bullet & \bullet & \star & \star \end{bmatrix}$$




$$\begin{bmatrix} \star & & \\ \bullet & \star & \\ \bullet & \star & \star \end{bmatrix}$$

➤ Two more steps [omitted - they involve no fill-in]

**Filled graph** = final graph including initial graph and all fill edges. Notation $G^F = (V, E^F)$. Note: $|E^F|$ measures the memory required for GE to solve the problem.

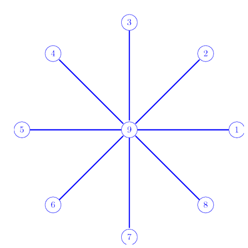


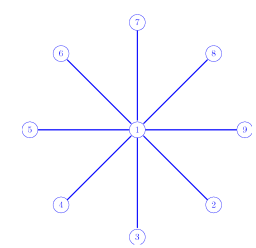



## *Gaussian Elimination. The fill path theorem*

➤ A Fill-path is a path between two vertices $i$ and $j$ in the graph of $A$ such that all vertices in the path, except the end points $i$ and $j$, are numbered less than $i$ and $j$.

✍₁₂ What are all the fill-paths for the two examples below



*THEOREM* [Rose-Tarjan] There is a fill-in in entry $(i, j)$ at the completion of the Gaussian elimination process if and only if, there exists a fill-path between $i$ and $j$.

➤ Example of application: Separating a graph $\equiv$ finding 3 sets of vertices: $V_1, V_2, S$ such that $V = V_1 \cup V_2 \cup S$ and $V_1$ and $V_2$ have no couplings. Labeling nodes of $S$ last prevents fill-ins between nodes of $V_1$ and $V_2$.

✍3 What are all the fill-edges for the previous examples (star graphs)

✍4 Fill-edges for: