SOLVING LINEAR ALGEBRAIC EQUATIONS

• The core computation engine for all circuit simulators is a linear solver. This is used even in EM-field solvers, mechanical engineering structural analysis packages, weather prediction software etc.

• In all such packages, dynamic non-linear analysis is done by translating the dynamic “circuit” into a non-linear “circuit” by numerical integration methods. This description is in turn translated into a linear “circuit” which is solved using a linear solver.

• So algorithms for solving linear algebraic equations are ubiquitous.

• Problem statement:
  Solve $A x = b$, where
  $A$ is a $n \times n$ real non-singular matrix
  $x$ is a $n \times 1$ matrix (column vector)
  $b$ is a $n \times 1$ matrix (column vector)

• In general there are two classes of methods:
  - Direct Methods (Gaussian Elimination, LU Decomposition, Crout’s method, Doolittle’s method).
- Direct methods give an exact solution in a finite number of steps, and are good for full matrices.

- Indirect (Iterative) Methods (Gauss-Jacobi, Gauss-Seidel, Successive Over Relaxation (SOR)).

- Indirect methods are iterative in nature, and may take a large number of steps to converge. Convergence depends heavily on the nature of the matrices (which depends on the nature of the circuit), so they are used in specific circuit instances when the matrix structure can be guaranteed to have the properties leading to quick convergence. In other instances, they may be a poor choice.

- Most simulators use direct methods since their computational complexity is bounded for arbitrary circuits.
GAUSSIAN ELIMINATION

- We use one equation to solve for one unknown in terms of other unknowns.

- Use this to replace a chosen unknown in the remaining equations

- Repeat (so that we have one less unknown in each iteration). This step is called **forward elimination**. Proceed until we have one equation in one unknown. Solve this.

- Repeatedly substitute to get the values of other unknowns. This is called **backward substitution**

- Example: Let $Ax = b$ be as shown below. Represent it as $A^{(1)}x^{(1)} = b^{(1)}$.

  $$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
  $$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
  $$\vdots$$
  $$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$
• To eliminate $x_1$, we write:

$$x_1 + \frac{a_{12}}{a_{11}}x_2 + \cdots + \frac{a_{1n}}{a_{11}}x_n = \frac{b_1}{a_{11}}$$

Alternately,

$$x_1 = \frac{b_1 - \sum_{j=2}^{n} a_{1j}x_j}{a_{11}}$$

• Now rewrite the remaining equations using this expression for $x_1$:

$$\frac{a_{21}}{a_{11}} \left( b_1 - \sum_{j=2}^{n} a_{1j}x_j \right) + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$

$$\vdots$$

$$\frac{a_{n1}}{a_{11}} \left( b_1 - \sum_{j=2}^{n} a_{1j}x_j \right) + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$

• We can rewrite this as:

$$(a_{22} - \frac{a_{12}a_{21}}{a_{11}})x_2 + (a_{23} - \frac{a_{12}a_{21}}{a_{11}})x_3 + \cdots = b_2 - \frac{a_{21}}{a_{11}}b_1$$

$$\vdots$$

$$(a_{n2} - \frac{a_{12}a_{n1}}{a_{11}})x_2 + \cdots = b_n - \frac{a_{n1}}{a_{11}}b_1$$

• We can write this as:

$$A^{(2)}x^{(2)} = b^{(2)}$$
where, (for $2 \leq i, j \leq n$)

$$a_{ij}^{(2)} = a_{ij} - \frac{a_{1j}a_{i1}}{a_{11}}$$

$$x^{(2)} = [x_2 \ x_3 \ \cdots \ x_n]^T$$

$$b_{i}^{(2)} = b_{i} - \frac{a_{i1}}{a_{11}} b_{1}$$

- When one more forward elimination step is performed, we get:

$$A^{(3)}x^{(3)} = b^{(3)}$$

where, (for $3 \leq i, j \leq n$)

$$a_{ij}^{(3)} = a_{ij}^{(2)} - \frac{a_{2j}^{(2)} a_{i2}^{(2)}}{a_{22}^{(2)}}$$

$$x^{(3)} = [x_3 \ x_4 \ \cdots \ x_n]^T$$

$$b_{i}^{(3)} = b_{i}^{(2)} - \frac{a_{i2}^{(2)}}{a_{22}^{(2)}} b_{2}^{(2)}$$

- Note that $l_{i2} = \frac{a_{i2}^{(2)}}{a_{22}^{(2)}}$ appears in the expression for $a_{ij}^{(3)}$ and $b_{i}^{(3)}$. We will utilize this property when we discuss LU Decomposition.
• We perform elimination in this way, until we get:

\[ A^{(n)}x^{(n)} = b^{(n)} \]

• Note that the forward elimination step has \( O(n^3) \) multiplications.
BACKWARD SUBSTITUTION

- Now collect the first equation of each step of the elimination:

\[ a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \]
\[ a_{22}x_2 + \cdots + a_{2n}x_n = b_2^{(2)} \]
\[ \vdots \]
\[ a_{nn}x_n = b_n^{(n)} \]

- This is an *Upper Triangular System* of equations

\[ Ux = \tilde{b} \]

where

\[ u_{ij} = \begin{cases} 
  a_{ij}^{(i)} & j \geq i \\
  0 & j < i 
\end{cases} \]

- So we can find \( x_n \) through \( x_1 \) as:

\[ x_n = \frac{\tilde{b}_n}{u_{nn}} \]
\[ x_{n-1} = \frac{\tilde{b}_{n-1} - u_{n-1,n}x_n}{u_{n-1,n-1}} \]
\[ x_1 = \frac{\tilde{b}_1 - \sum_{j=2}^{n} u_{1j} x_j}{u_{11}} \]

- Note that the back substitution step has \( \mathcal{O}(n^2) \) multiplications.

- The overall process, however, has \( \mathcal{O}(n^3) \) multiplications due to the forward elimination step.

- This gives us a solution algorithm, but the problem is that if \( b \) changes, then we have to redo the entire process of elimination and substitution.

- Since \( b \) comprises the source or forcing terms, we would need to solve GE each time for the different excitations. How do we avoid this?
LU DECOMPOSITION

- At the $j^{th}$ step of GE, we multiply the $j^{th}$ equation by $l_{ij}$ and subtract it from the $i^{th}$ equation, where, as we saw earlier,

$$l_{ij} = \frac{a_{ij}^{(j)}}{a_{jj}^{(j)}}$$

So,

$$a_{ik}^{(j+1)} = a_{ik}^{(j)} - l_{ij}a_{jk}^{(j)}$$

- We can represent this as a matrix multiplying $A^{(j)}$:

$$(L_j^{-1})A^{(j)} = A^{(j+1)}$$

where

$$(L_j^{-1}) = I - l^{(j)}e_j^T$$

and $l^{(j)}$ is a column vector given by:

$$l_i^{(j)} = \begin{cases} 0 & i \leq j \\ l_{ij} & i > j \end{cases}$$
\[ e_j = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1(j^{th} \text{row}) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \]

- Therefore, \( L_j^{-1} \) has 1's along its diagonal, and the \( j^{th} \) column has entries \(-l_{ij}^{(j)}\) below the diagonal. All other entries are 0.

- We can rewrite the GE operation as:

\[ U = (L_n^{-1})(L_{n-1}^{-1}) \cdots (L_1^{-1})A \]

- Note that \( L_j = (L_j^{-1})^{-1} = I + l^{(j)}e_j^T \). Verifying this, \((I - l^{(j)}e_j^T)(I + l^{(j)}e_j^T) = I + l^{(j)}e_j^T - l^{(j)}e_j^T + l^{(j)}[e_j^Tl^{(j)}]e_j^T = I\). The term in square braces is 0.

- Further, for \( i < j \), \( L_iL_j = (I + l^{(i)}e_i^T)(I + l^{(j)}e_j^T) = I + l^{(i)}e_i^T + l^{(j)}e_j^T \), by a similar argument as above.

- Hence \( L_1L_2 \cdots L_n = L \), a Lower Triangular Matrix. In particular, it’s diagonal elements are 1.

- Hence \( A = LU \), with \( l_{ii} = 1 \)
• Note that $L$ and $U$ depend on $A$, so they **do not** depend on the excitation matrix $b$. Hence, once we find $L$ and $U$ we can solve for arbitrary excitations ($O(n^2)$ work) without re-doing the forward elimination work (which is $O(n^3)$).

• This is very important for dynamic simulations, where the RHS ($b$ matrix) which contains the sources or forcing terms, may change.
LU DECOMPOSITION IN PRACTICE

• As we saw, \( A = LU \).

• Recall we are solving for \( Ax = b \equiv LUx = b \).

• Step 1: Let \( y = Ux \). Then \( Ly = b \). Solve for \( y \). This requires \( \frac{n(n-1)}{2} \) multiplications (since diagonal entries of \( L \) are 1) and is called the forward substitution step.

• Step 2: Now solve for \( x \) in \( Ux = y \). This requires \( \frac{n(n+1)}{2} \) (since diagonal entries of \( U \) are arbitrary) multiplications and is called the backward substitution step.
COMPUTING L AND U FROM A

- We know that $A = LU$

- Therefore, by multiplying L and U, we can get:

$$a_{ij} = \begin{cases} 
\sum_{k=1}^{i} l_{ik} u_{kj} & i \leq j \\
\sum_{k=1}^{j} l_{ik} u_{kj} & i > j
\end{cases}$$

- This gives us a way of computing $l_{ij}$ and $u_{ij}$ entries:

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \quad i \leq j$$

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}}{u_{jj}} \quad i > j$$

- Note that $l_{ii} = 1$ hence the $u_{ij}$ equation has a slightly different structure than the $u_{ij}$ equation.
ORDER OF COMPUTING L AND U ELEMENTS

- There are two possible orders in computing L and U elements.

- The dependencies for any given element are as shown below. In this figure, the entry being computed is marked by the “X”, and the shaded region represents the elements that the computed element depends upon.
RELATION BETWEEN STA AND NA

• Recall the STA equations are

\[
\begin{bmatrix}
K_i & -K_v & 0 \\
0 & I & -A^T \\
A & 0 & 0
\end{bmatrix}
\begin{bmatrix}
i \\
v \\
e
\end{bmatrix}
= \begin{bmatrix}
S \\
0 \\
0
\end{bmatrix}
\]

• By multiplying the first set of equations by \(K_i^{-1}\) and then by multiplying the (new) first set of equations by \(-A\) and adding to the third set of equations, we can get

\[
\begin{bmatrix}
I & -K_i^{-1}K_v & 0 \\
0 & I & -A^T \\
0 & AK_i^{-1}K_v & 0
\end{bmatrix}
\begin{bmatrix}
i \\
v \\
e
\end{bmatrix}
= \begin{bmatrix}
K_i^{-1}S \\
0 \\
-AK_i^{-1}S
\end{bmatrix}
\]

• Finally, multiplying the second set of equations by \(-AK_i^{-1}K_v\) and adding to the third set of equations, we get

\[
\begin{bmatrix}
I & -K_i^{-1}K_v & 0 \\
0 & I & -A^T \\
0 & 0 & AK_i^{-1}K_vA^T
\end{bmatrix}
\begin{bmatrix}
i \\
v \\
e
\end{bmatrix}
= \begin{bmatrix}
K_i-1S \\
0 \\
-AK_i^{-1}S
\end{bmatrix}
\]

• Note that the third set of equations now is of the form \(Y_ne = IS\) where \(Y_n = AK_i^{-1}K_vA^T\) and \(IS = -AK_i^{-1}S\).
• In other words, by linear transformations, we can get the NA equations from the STA equations. Therefore NA is obtained by a particular sequence of operations on the STA formulation.

• STA is most efficient, therefore, since NA corresponds to a particular transformation of the STA formulation (which is most “natural” and resembles equations written by hand)

• However, practical considerations dictate the common use of NA-like techniques. In particular, STA requires complex programming techniques to exploit sparsity. Also, multiplication by factors like -1, 1 need to be handled carefully.
IMPROVING ACCURACY – “PIVOTING”

- When diagonal elements are too small relative to other elements, significant numerical error may result.

- Consider an MNA matrix on which we perform LU Decomposition. After 2 steps of LUD it becomes, say,

\[
\begin{bmatrix}
X & X & 0 & 0 & 0 & 0 \\
0 & X & 0 & 0 & X & 0 \\
0 & 0 & \frac{1}{R} & \frac{-1}{R} & 1 & 0 \\
0 & 0 & \frac{-1}{R} & \frac{1}{R} & 0 & 1 \\
0 & 0 & X & 0 & X & 0 \\
0 & 0 & 0 & X & X & 0
\end{bmatrix}
\]

- After a further step, it becomes:

\[
\begin{bmatrix}
X & X & 0 & 0 & 0 & 0 \\
0 & X & 0 & 0 & X & 0 \\
0 & 0 & \frac{1}{R} & \frac{-1}{R} & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & X & X & 0 \\
0 & 0 & 0 & X & X & 0
\end{bmatrix}
\]

- For this step, \( l_{i4} = \frac{d_{i4}^{(a)}}{d_{44}^{(a)}} \) which is unbounded.
• Pictorially,

After Step 2

After Step 3

• How do we fix this problem? By a technique called **pivoting**

• Pivoting involves interchanging rows and/or columns so that we get a non-zero element in the position $(i,i)$. For example, in the matrix below, we could swap rows 1 and 2 to eliminate the zero diagonal element of the first row.

$$
\begin{bmatrix}
0 & 1 & 1 \\
X & X & 0 \\
X & X & 0 \\
\end{bmatrix} \rightarrow \begin{bmatrix}
X & X & 0 \\
0 & 1 & 1 \\
X & X & 0 \\
\end{bmatrix}
$$

• Lets see an example of the problems if we did not perform pivoting. Suppose we are solving

$$
\begin{bmatrix}
1.25 \times 10^{-4} & 1.25 \\
12.5 & 12.5 \\
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2 \\
\end{bmatrix} = \begin{bmatrix}
6.25 \\
75 \\
\end{bmatrix}
$$
• The solution of this to 4 decimal places is \(x_1 = 1.0001, x_2 = 5.0000\).

• Assume 3-digit floating point arithmetic. The first step of GE gives, recalling that

\[
a^{(2)}_{ij} = a^{(1)}_{ij} - \frac{a^{(1)}_{ij} a^{(1)}_{i1}}{a^{(1)}_{11}}
\]

\[
b^{(2)}_i = b^{(1)}_i - \frac{a^{(1)}_{i1} b^{(1)}_1}{a^{(1)}_{11}}
\]

Hence we get

\[
\begin{bmatrix}
1.25 \times 10^{-4} & 1.25 \\
0 & -1.25 \times 10^5
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
= \begin{bmatrix}
6.25 \\
-6.25 \times 10^5
\end{bmatrix}
\]

• So, with GE we get \(x_1 = 0, x_2 = 5\).

• Now lets do the same with row interchanging. The first step of GE yields

\[
\begin{bmatrix}
12.5 & 12.5 \\
0 & 1.25
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
= \begin{bmatrix}
75 \\
6.25
\end{bmatrix}
\]

• Therefore, using GE with row interchange, however, we get \(x_1 = 1, x_2 = 5\), as desired.

• Why did this occur? Because without row interchange, \(a_{11}\) was too small relative to other numbers.
• Pivoting can be of different styles. Let's see these next
PIVOTING TECHNIQUES

- **Partial Pivoting:** This involves row interchange only. In this method, we choose a pivot \( r \) which is the smallest integer such that

\[
|a_{rk}^{(k)}| = \max_{j=k, \ldots, n} |a_{jk}^{(k)}|
\]

- The search space for partial pivoting is showed as the shaded region below

- **Complete Pivoting:** This involves row as well as column interchange. In this method, we choose a pivot \( r \) and \( s \) which are the smallest integers such that

\[
|a_{rs}^{(k)}| = \max_{i=k, \ldots, n; j=k, \ldots, n} |a_{ij}^{(k)}|
\]
• The search space for complete pivoting is showed as the shaded region below

• **Threshold Pivoting**: This method is applied conditionally. The two flavors of threshold pivoting are:
  
  – Apply partial pivoting only if
    
    \[ |a_{kk}^{(k)}| < \varepsilon_p |a_{rk}^{(k)}| \]

  – Apply complete pivoting only if
    
    \[ |a_{kk}^{(k)}| < \varepsilon_p |a_{rs}^{(k)}| \]

• Note that in threshold pivoting, we are comparing the diagonal element with the largest element in the corresponding search region as a criterion for applying pivoting.
In SPICE, the parameters abstol and reltol are parameters that affect the choice of threshold for pivoting.
ERROR MECHANISMS

- There are two kinds of accuracy problems we may face while solving a set of linear equations
  - Ill-conditioned matrix ("almost" singular matrix). There is no way to avert this problem since the circuit dictates the MNA matrix.
  - Problems due to numerical stability of the solution method. This can be averted by a technique called pivoting.

- Ill-conditioning can be detected by means of norms.

- **Vector Norms:** There are several such norms.

\[
L_1 : ||X||_1 = \sum_{i=1}^{n} |x_i|
\]

\[
L_2 : ||X||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}
\]

\[\vdots\]

\[
L_n : ||X||_n = \left(\sum_{i=1}^{n} |x_i^n|\right)^{\frac{1}{n}}
\]

\[\vdots\]
\[ L_\infty : \|X\|_\infty = \max_{1 \leq i \leq n} |x_i| \]

- **Matrix Norms** are related to vector norms, and computed as

\[ ||A|| = \max_{X \neq 0} \frac{||AX||}{||X||} \]

- The various matrix norms are

\[ L_1 : ||A||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{ij}| \]

\[ L_2 : ||A||_2 = (\text{largest eigenvalue of } A^T A)^{\frac{1}{2}} \]

\[ : \]

\[ L_\infty : ||A||_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}| \]

- Note that the \( L_1 \) matrix norm is the column with the largest sum of the absolute values of all its elements.

- Likewise the \( L_\infty \) matrix norm is the row with the largest sum of the absolute values of all its elements.

- Also, by definition, we have

\[ ||Ax|| \leq ||A|| ||x|| \]
DETECTING ILL-CONDITIONING

- Suppose we are given $A$, $b$ and $x$ such that $Ax = b$.

- By definition, $||b|| \leq ||A|| \cdot ||x||$

- **If we perturb $b$:** $b \rightarrow b + \delta b$

  $A(x + \delta x) = b + \delta b$

  $A\delta x = \delta b$

  $||\delta x|| \leq ||A^{-1}|| \cdot ||\delta b||$

  $\frac{||\delta x||}{||x||} \leq ||A|| \cdot ||A^{-1}|| \cdot \frac{||\delta b||}{||b||} = K(A) \frac{||\delta b||}{||b||}$

  where

  $K(A) = ||A|| \cdot ||A^{-1}||$

- $K(A)$ is called the **condition number** of $A$. It represents the “amplification” of a perturbation in $x$ as a result of a perturbation in $b$. A large condition number indicates an ill-conditioned problem and is undesirable.

- **If we perturb $A$:** $A \rightarrow A + \delta A$

  $(A + \delta A)(x + \delta x) = b$
\[ A\delta x + \delta A(x + \delta x) = 0 \]
\[ ||\delta x|| \leq ||A^{-1}|| ||\delta A|| ||x + \delta x|| \]
\[ \frac{||\delta x||}{||x + \delta x||} \leq ||A|| ||A^{-1}|| \frac{||\delta A||}{||A||} = K(A) \frac{||\delta A||}{||A||} \]

- So perturbations in either \( A \) or \( b \) result in a perturbation in \( x \) which “magnifies” the perturbation in \( A \) or \( b \) by \( K(A) \).
SCALING AND EQUILIBRATION

- We can **scale columns** or **equilibrate rows** in order to obtain more accurate solutions by allowing better pivoting choices.

- Let $Ax = b$.

- Rescale $x$ by using $\tilde{x} = D_1^{-1}x$, where $D_1 = \text{diagonal}(\alpha_1, \ldots, \alpha_n)$

- Note that this results in scaled columns. This allows us to effectively rescale unknowns to reflect the different units of these unknowns (e.g. volts and micro-amperes).

- Equilibrate rows by using $D_2AD_1\tilde{x} = D_2b = \tilde{b}$ where $D_2 = \text{diagonal}(\beta_1, \ldots, \beta_n)$

- Note that equilibrating is done so as to ensure that row elements are of similar magnitude. For this, choose $D_2$ such that $\max_j |a_{ij}| < 1$

- Consider the following problem:

$$\begin{bmatrix} 1 & 10000 \\ 1 & 0.0001 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 10000 \\ 1 \end{bmatrix}$$
• With partial pivoting, we get a solution \( \overline{x_2} = 1, \overline{x_1} = 0 \) (to 2 decimal places)

• If we equilibrate the rows, then we have to solve

\[
\begin{bmatrix}
0.0001 & 1 \\
1 & 0.0001
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
=
\begin{bmatrix}
1 \\
1
\end{bmatrix}
\]

• Which gives a solution (partial pivoting, 2 decimal places) of \( \overline{x_1} = 1, \overline{x_2} = 1 \) as desired.
EFFECTIVENESS OF SCALING AND EQUILIBRATION

- **Theorem:** Let $\bar{x}$ and $\bar{x}'$ be solutions of $Ax = b$ and $(D_2 AD_1)x' = (D_2 b)$, where $D_1$ and $D_2$ are powers of 2 (so there is no roundoff error on a computer) and the same pivoting order is used in solving for $x$ and $x'$. Then $\bar{x}$ and $\bar{x}'$ differ only in their exponents (i.e. $\bar{x} = D_1 \bar{x}'$).

- The implication of this is that the only function of scaling and equilibration is to affect the choice of pivoting elements.

SPARSE MATRIX TECHNOLOGY

- Suppose we are solving a sparse system. In that case, there will be many computations of the kind $0 \times 0 + 0$, in a typical LUD procedure.

- Sparse matrix solution techniques would recognize such a situation and simply not perform the computation.

- For large sparse matrices this in fact saves significant computation (and storage) resources.

- Before starting the solutions process, a list of computations which are of the type $*0$, $+0$, $*1$, $*(-1)$ is made. This list is used during the solution process, avoiding such computations altogether.

- For example, in a system of 1000 equations, full matrix based GE would require $O(n^3)$ computations $\approx 10^9$ flops. This would require $10^3$ seconds on a computer which can do 1 Mflop/sec. Storage would be about $10^6$ words.

- The same problem, using MNA with 3 non-zero elements per row, would require approximately 6000 words. Assuming sparse matrix GE complexity of
$O(n^{1.1}) - O(n^{1.5})$ (this is determined experimentally) this would require approximately 0.001 - 0.03 sec.

- The problem is that the LUD algorithm needs to be coded more carefully to exploit sparsity. Also, sparsity gets affected by the order of doing elimination. So care has to be taken that the matrix does not lose sparsity during the elimination.

- Let us next look into the techniques utilized by sparse matrix solution techniques.
GENERAL RULES FOR SOLVING SPARSE MATRICES

• Avoid storing zero elements. In practice, this is done by using a linked-list or pointer based data structure.

• Avoid trivial operations
  - \( 0 \times x = 0 \)
  - \( 0 + x = x \)
  - \( 1 \times x = x \)

• Zeros are of two kinds
  - Structural zeros (always zero)
  - Numerical zeros (structural non-zeros whose value happens to be zero at some point in the computation). Such zeros occur as a result of the elimination calculations. We need to store them as though they were non-zeros since we can’t know that they will change to non-zeros later
• Don’t lose sparsity
   Avoid:
   \[
   \begin{bmatrix}
   x & x & x & x & x \\
   x & x & 0 & 0 & 0 \\
   x & 0 & x & 0 & 0 \\
   x & 0 & 0 & x & 0 \\
   x & 0 & 0 & 0 & x
   \end{bmatrix}
   \xrightarrow{\text{After LUD}}
   \begin{bmatrix}
   x & x & x & x & x \\
   x & x & x & x & x \\
   x & x & x & x & x \\
   x & x & x & x & x \\
   x & x & x & x & x
   \end{bmatrix}
   = L/U
   \]

   Instead, pivot first to get
   \[
   \begin{bmatrix}
   x & 0 & 0 & 0 & x \\
   0 & x & 0 & 0 & x \\
   0 & 0 & x & 0 & x \\
   0 & 0 & 0 & x & x \\
   x & x & x & x & x
   \end{bmatrix}
   \xrightarrow{\text{After LUD}}
   \begin{bmatrix}
   x & 0 & 0 & 0 & x \\
   0 & x & 0 & 0 & x \\
   0 & 0 & x & 0 & x \\
   0 & 0 & 0 & x & x \\
   x & x & x & x & x
   \end{bmatrix}
   = L/U
   \]

• The “new” \(x\)'s are referred to as \textbf{fill-ins}. Our goal is to minimize these fill-ins.

• Example:
   \[
   \begin{bmatrix}
   1 & 2 \\
   1 & 0
   \end{bmatrix}
   \xrightarrow{\text{After LUD}}
   \begin{bmatrix}
   1 & 2 \\
   1 & -2
   \end{bmatrix}
   \]

• The “-2” entry above is a fill-in
CHOOSING A PIVOT - MARKOWITZ CRITERION

- To minimize the number of fill-ins is an NP-complete problem.

- So we use a “greedy” algorithm which uses the Markowitz criterion. Actually this criterion minimizes multiplications.

- Assume we are attempting to find the \( k^{th} \) pivot (for \( A^{(k)} \)). Let \( r_i \) be the number of non-zeros for the \( i^{th} \) row, and \( c_j \) be the number of non-zeros for the \( j^{th} \) column.

- The Markowitz product for \( A_{ij}^{(k)} \) is \( (r_i-1)(c_j-1) \). This is the maximum number of fill-ins due to the choice of the \((i, j)\) pivot.

- \( (r_i-1) \) is the number of fill-ins per row that gets filled in (since the first element is eliminated).

- \( (c_j-1) \) is the number of filled rows (since elements with 0’s in their column need not be touched, and also the first element is eliminated).
• The number of multiplications required in GE to pivot on \((i, j)\) is \(r_i(c_j - 1)\) (since \(b\) needs to be updated as well).

• So the heuristic is to choose the pivot element with the smallest Markowitz product. This can be done apriori since the values of the excitation vector may change, but the circuit topology is unchanged. This is very desirable.

• Several modifications are possible:
  – Ties can be broken by choosing the element with the largest magnitude.
  – Use threshold pivoting
  – Try threshold test \textbf{after} choosing the smallest Markowitz product.

• In general, NA matrices tend to be diagonally dominant, hence the modifications which involve choosing good pivots are less important. For STA matrices, however, the choice of good pivot must be combined with the Markowitz product based decision.
**MOTIVATION FOR MARKOWITZ CRITERION**

Case a)

\[
\begin{array}{c|c}
  j & k \\
  \hline
  j & X & X \\
  i & X & 0 \\
\end{array}
\]

Will \(a_{ik}^{(j+1)}\) get filled in?

\[
a_{ik}^{(j+1)} = a_{ik}^{(j)} - a_{jk}^{(j)} \frac{d_{ik}^{(j)}}{d_{jk}^{(j)}}
\]

\[
a_{ik}^{(j+1)} = 0 - X \frac{X}{X} \equiv \text{YES!}
\]

Case c)

\[
\begin{array}{c|c}
  j & k \\
  \hline
  j & X & 0 \\
  i & X & 0 \\
\end{array}
\]

Will \(a_{ik}^{(j+1)}\) get filled in?

\[
a_{ik}^{(j+1)} = 0 - 0 \frac{X}{X} \equiv \text{NO!}
\]

Case b)

\[
\begin{array}{c|c}
  j & k \\
  \hline
  j & X & X \\
  i & 0 & 0 \\
\end{array}
\]

Will \(a_{ik}^{(j+1)}\) get filled in?

\[
a_{ik}^{(j+1)} = 0 - X \frac{0}{X} \equiv \text{NO!}
\]

Case d)

\[
\begin{array}{c|c}
  j & k \\
  \hline
  j & X & 0 \\
  i & 0 & 0 \\
\end{array}
\]

Will \(a_{ik}^{(j+1)}\) get filled in?

\[
a_{ik}^{(j+1)} = 0 - 0 \frac{0}{X} \equiv \text{NO!}
\]
• So only case a) gets a fill-in. It is possible that there is no fill-in (if \( a_{ik}^{(j)} \) was “X” to start with).

• Markowitz criterion computes the **maximum** number of fill-ins

• The maximum number of fill-ins is therefore \((r_i - 1)(c_j - 1)\).