Systems of Linear Equations

The purpose of computing is insight, not numbers.

Richard Wesley Hamming

Fall 2010

Topics to Be Discussed

- **This is a long unit and will include the following important topics:**
  - **Solving systems of linear equations – Gaussian elimination**
  - **Pivoting**
  - **LU-decomposition**
  - **Iterative methods (Jacobi and Gauss-Seidel)**
  - **Iterative Refinement**
  - **Matrix Inversion**
  - **Determinant**

Problem Description: 1/2

- **Solving systems of linear equations is one of the most important tasks in numerical methods.**
- **The i-th equation ($1 \le i \le n$) is $a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + \ldots\ldots + a_{in}x_n = b_i$, where a_{i1}, a_{i2}, a_{i3}, ……, a_{in} and b_i are known values, and the x_i's are unknowns to be solved from the n linear equations.**

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots\ldots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots\ldots + a_{2n}x_n &= b_2 \\
&\ldots\ldots\ldots\ldots \\
a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots\ldots + a_{nn}x_n &= b_n
\end{aligned}
$$

Problem Description: 2/2

- **A system of linear equations is usually represented by matrices, $\mathbf{A}=[a_{ij}]_{n\times n}$ the coefficient matrix, $[b_i]_{n\times 1}$ the constant matrix, and $[x_i]_{n\times 1}$ the unknown matrix.**

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{n1} \\ a_{21} & a_{22} & \cdots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Methods to Be Discussed

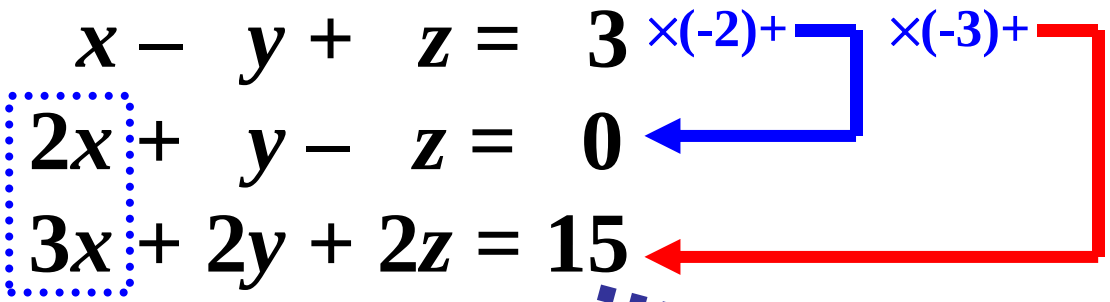
- **Methods for solving systems of linear equations are of two types, *direct* and *iterative*.**
- ***Direct* methods go through a definite procedure to find the solution. Elimination and decomposition are the two major approaches.**
- ***Iterative* methods, which is similar to solving non-linear equations, find the solution iteratively.**

Gaussian Elimination: 1/7

- **Suppose all conditions are ideal (*i.e.*, no errors will occur during the computation process).**
- **Gaussian elimination is very effective and easy to implement.**
- **Idea 1: For every *i*, use the *i*-th equation to eliminate the unknown x_i from equations *i*+1 to *n*. This is the *elimination* stage.**
- **Idea 2: After the elimination stage, a *backward substitution* is performed to find the solutions.**

Gaussian Elimination: 2/7

- **The following is a simple elimination example. Use equation 1 to eliminate variable *x* in equations 2 and 3.**



$$x - y + z = 3$$
$$3y - 3z = -6$$
$$5y - z = 6$$

***x* is eliminated from equations 2 and 3**

Gaussian Elimination: 3/7

- **Use equation 2 to eliminate *y* in equation 3.**
- **After elimination, the system is *upper triangular*!**

$$\begin{aligned} x - y + z &= 3 \\ 3y - 3z &= -6 \\ 5y - z &= 6 \end{aligned}$$

$\times(-5/3)+$

$$\begin{aligned} x - y + z &= 3 \\ 3y - 3z &= -6 \\ 4z &= 16 \end{aligned}$$

***y* is eliminated from equation 3**
Equation 2 only has *z*!

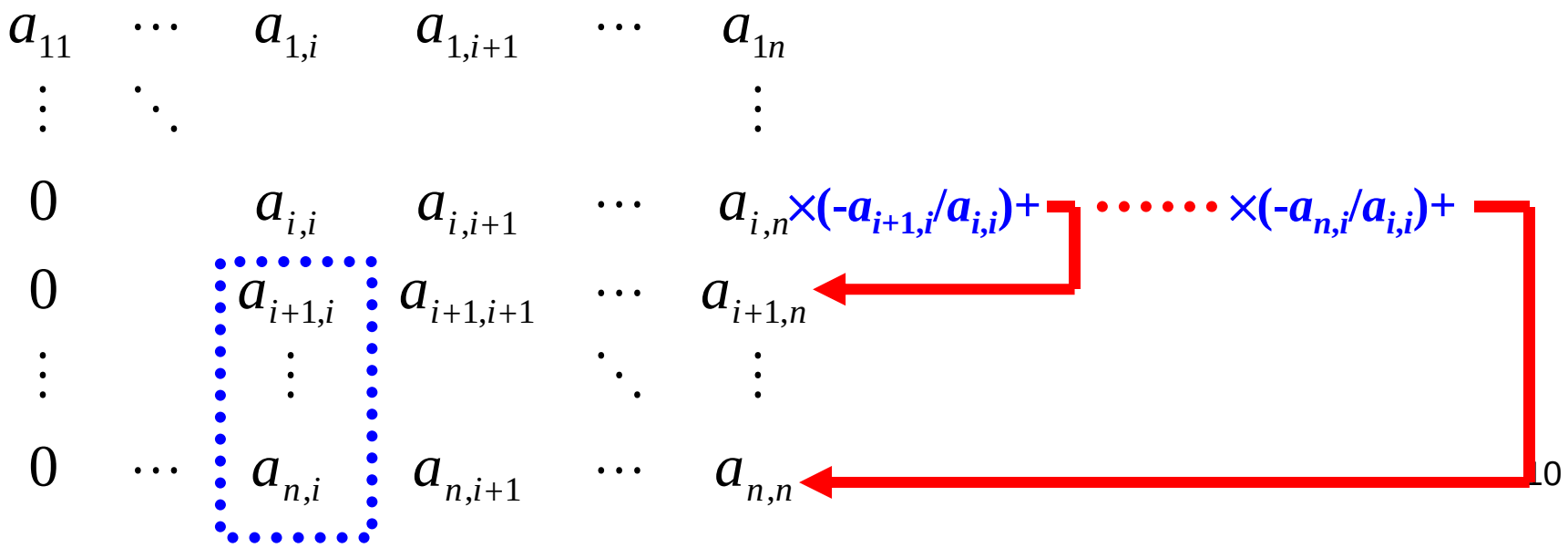
Gaussian Elimination: 4/7

- **Now we can solve for *z* from equation 3.**
- **Plug *z* into equation 2 to solve for *y*.**
- **Then, plug *y* and *z* into equation 1 to solve for *x*.**
- **This is *backward substitution*!**

$$
\begin{aligned}
x - y + z &= 3 \\
3y - 3z &= -6 \\
4z &= 16
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
x &= 3+y-z \\
y &= (-6+3z)/3 \\
z &= 16/4
\end{aligned}
$$

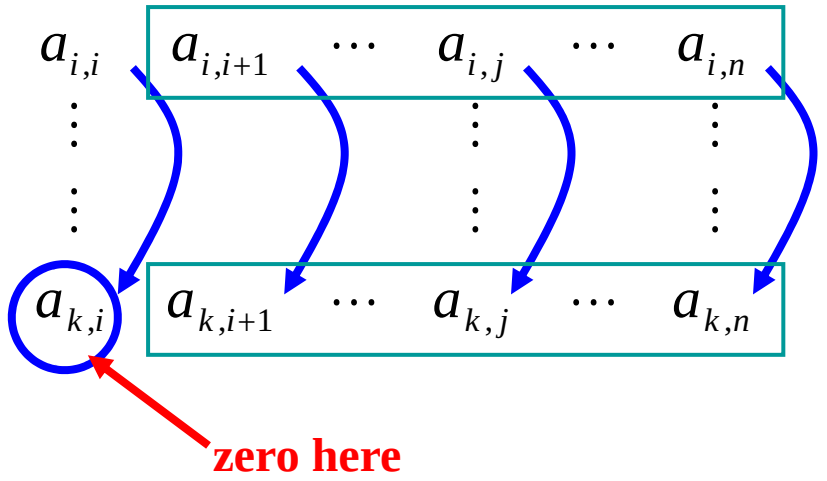
Gaussian Elimination: 5/7

- **Each entry of row i and b_i is multiplied by $-a_{k,i}/a_{i,i}$ and added to row k and b_k for all k with $i+1 \le k \le n$.**
- **In this way, all entries on column i below $a_{i,i}$ are eliminated (*i.e.*, zero). Do the same for b_i's.**



Gaussian Elimination: 6/7

- **Gaussian Elimination uses row *i* (*i.e.*, $a_{i,i}$) to eliminate $a_{k,i}$ for $k > i$.**
- **Thus, all entries below $a_{i,i}$ become zero!**




```
DO i = 1, n-1                          ! using row i, i.e., a(i,*)
  DO k = i+1, n                        ! we want to eliminate a(k,i)
    S = -a(k,i)/a(i,i)                 ! compute the multiplier
    DO j = i+1, n                      ! for each entry on row k
      a(k,j) = a(k,j) + S*a(i,j)       !   update its value
    END DO
    a(k,i) = 0                         ! set a(k,i) to 0
    b(k) = b(k) + S*b(i)               ! don't forget to update b(k)
  END DO
END DO
```

Gaussian Elimination: 7/7

- **After elimination, the equations become *upper triangular*, *i.e.*, all lower diagonal entries being 0's.**

All Zeros

$$\begin{bmatrix} a_{11} & \cdots & a_{1,i} & a_{1,i+1} & \cdots & a_{1n} \\ \vdots & \ddots & & & & \vdots \\ 0 & & a_{i,i} & a_{i,i+1} & \cdots & a_{i,n} \\ 0 & & 0 & a_{i+1,i+1} & \cdots & a_{i+1,n} \\ \vdots & & \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & & a_{n,n} \end{bmatrix}$$

- **Equation *i* only has variables x_i, x_{i+1}, …, x_n:**

$$a_{i,i}x_i + a_{i,i+1}x_{i+1} + \cdots + a_{i,n}x_n = b_i$$

Backward Substitution

- Equation n is $a_{n,n}x_n = b_n$, and hence $x_n = b_n/a_{n,n}$.
- Equation n-1 is $a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = b_{n-1}$, and $x_{n-1} = (b_{n-1} - a_{n-1,n}x_n)/a_{n-1,n-1}$
- Equation i is $a_{i,i}x_i + a_{i,i+1}x_{i+1} + \cdots + a_{i,n}x_n = b_i$ and x_i is computed as:

$$x_i = \frac{1}{a_{i,i}}\left(b_i - \sum_{k=i+1}^{n} a_{i,k}x_k\right)$$

```
DO i = n, 1, -1              ! going backward, row i
  S = b(i)                   ! initialize S to b(i)
  DO k = i+1, n              ! sum all terms to the right
    S = S - a(i,k)*x(k)      !   of a(i,i)
  END DO
  x(i) = S/a(i,i)            ! compute x(i)
END DO
```

Efficiency Issues: 1/4

- **How to determine the efficiency of Gaussian elimination?**
  - **We count the number of multiplications and divisions, since they are slower than additions and subtractions.**
  - **As long as we know the dominating factor, we know the key of efficiency.**
  - **Since divisions are not used frequently, counting multiplications would be good enough.**

Efficiency Issues: 2/4

- Fixing *i*, the inner-most *j* loop executes *n-i* times, and uses *n-i* multiplications. One more is needed to update b_k. Therefore, the total is *n-i*+1.
- Since *k* goes from *i*+1 to *n* and the inner-most *j* loop executes *n-i* times, there are (*n-i*)(*n-i*+1) multiplications.
- Since *i* goes from 1 to *n*-1, the total is $\sum_{i=1}^{n-1}(n-i)(n-i+1)$

```
DO i = 1, n-1
  DO k = i+1, n
    S = -a(k,i)/a(i,i)
    DO j = i+1, n
      a(k,j) = a(k,j) + S*a(i,j)
    END DO
    a(k,j) = 0
    b(k) = b(k) + S*b(i)
  END DO
END DO
```

This `j` loop uses *n-i* *'s

One more * here

Thus, one `i` iteration uses *n-i*+1 *'s

Efficiency Issues: 3/4

- **The following can be proved with induction:**

$$1+2+\cdots n=\frac{1}{2}n(n+1)$$

$$1^2+2^2+3^2+\cdots+n^2=\frac{1}{6}\left(2n^3+3n^2+n\right)$$

- **Therefore, we have (verify yourself):**

$$\sum_{i=1}^{n-1}(n-i)(n-i+1)=\sum_{i=1}^{n-1}(n-i)^2+\sum_{i-1}^{n-1}(n-i)=\frac{1}{3}\left(n^3-n\right)$$

- **This is an $O(n^3)$ algorithm (*i.e.*, number of multiplications proportional to n^3).**

Efficiency Issues: 4/4

- **In the *backward substitution* step, since we need to compute the sum $a_{i,i+1}x_{i+1}+a_{i,i+2}x_{i+2}+\ldots+a_{i,n}x_n$, $n-i$ multiplications are needed.**
- **Since i runs from $n-1$ down to 1, the total number of multiplications is $\sum_{i=1}^{n-1}(n-i)=\frac{1}{2}n(n-1)$**
- **In summary, the total number of multiplications used is dominated by the elimination step, which is proportional to $n^3/3$, or $O(n^3)$!**
- **Exercise: *How many divisions are needed in the elimination and the backward substitution stages?***

A Serious Problem

- **There is a big problem. When eliminating $a_{k,i}$ using $a_{i,i}$, $-a_{k,i} / a_{i,i}$ is computed.**
- **What if $a_{i,i} \approx 0$? Division-by-zero or overflow in $-a_{k,i}/a_{i,i}$ or truncation in $(-a_{k,i}/a_{i,i})\times a_{i,j}+a_{k,j}$ may occur because of $(-a_{k,i}/a_{i,i})\times a_{i,j} >> a_{k,j}$! Why?**
- **Therefore, the use of Gaussian elimination is risky and some *modifications* would be needed.**
- **However, if the following holds (*i.e.*, *diagonal dominant*), Gaussian elimination works fine.**

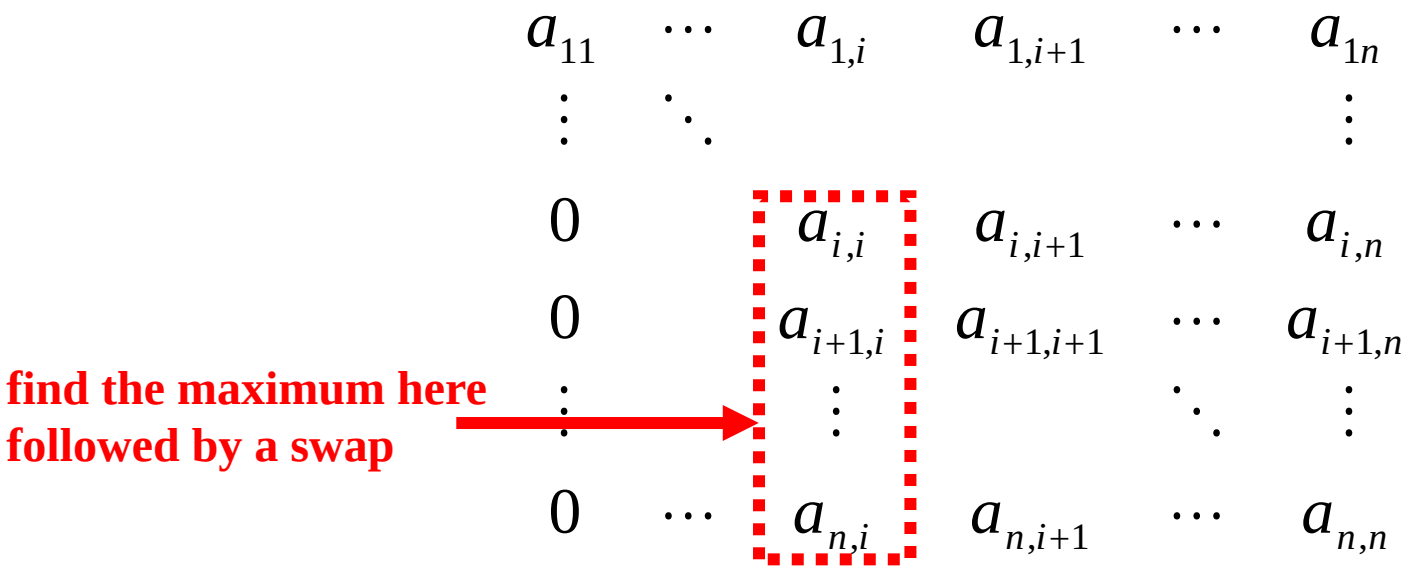
$$|a_{i,i}| > \sum_{j=1, j\neq i}^{n} |a_{i,j}| \quad \textbf{(for every } i\textbf{)}$$

Partial Pivoting: 1/6

- **To overcome the possible *x*/0 issue, one may use an important technique: *pivoting*.**
- **There are two types of pivoting, *partial* (or *row*) and *complete*. For most cases, *partial pivoting* usually works efficiently and accurately.**
- **Important Observation: *Swapping two equations does not change the solutions!***
- **If $a_{i,i} \approx 0$, one may swap some equation k with equation i so that the new $a_{i,i}$ would not be zero.**
- **But, which k should be swapped?**

Partial Pivoting: 2/6

- **One of the best candidates is the $a_{k,i}$ such that $|a_{k,i}|$ is the maximum of $|a_{i,i}|$, $|a_{i+1,i}|$, …, $|a_{n,i}|$.**
- **Why? If equation k with maximum $|a_{k,i}|$ is swapped to equation i, then, all $|a_{j,i}/a_{i,i}| \le 1$ for $i \le j \le n$.**



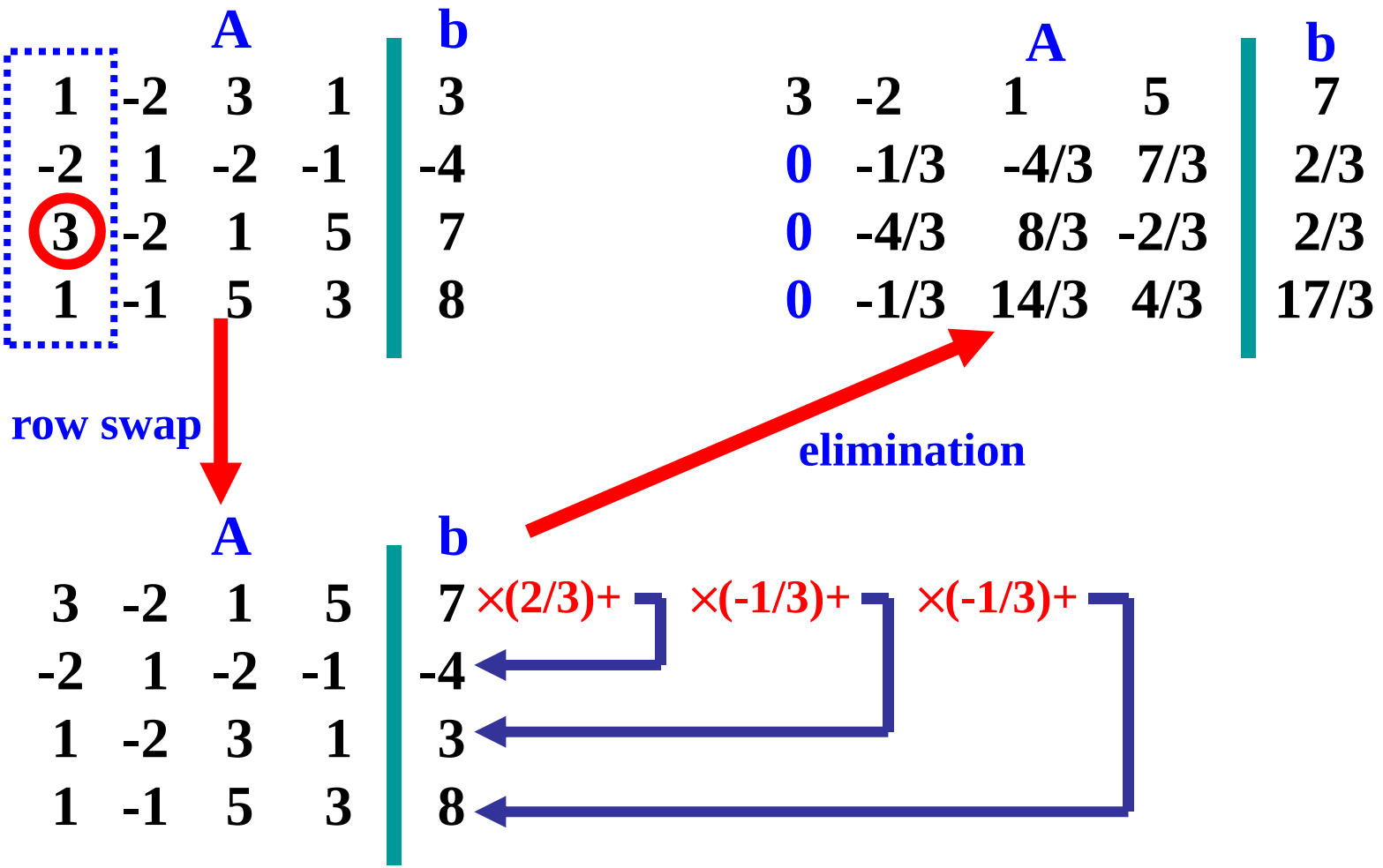
Partial Pivoting: 3/6

- **The original Gaussian elimination is modified to include pivoting: find the maximum $|a_{k,i}|$ of $a_{i,i}$, $a_{i+1,i}$, …, $a_{n,i}$, and do a row swap, including b_i and b_k. The remaining is the same!**

```
DO i = 1, n-1                    ! going through rows i+1 to n
  Max = i                        ! assume |a(i,i)| is the max
  DO k = i+1, n                  ! find the largest in Max
    IF (ABS(a(Max,i)) < ABS(a(k,i)) Max = k
  END DO
  DO j = i, n                    ! swap row Max and row i
     swap a(Max,j) and a(i,j)
  END DO
  swap b(Max) and b(i)           ! don't forget to swap b(i)
  … do the elimination step …
END DO
```

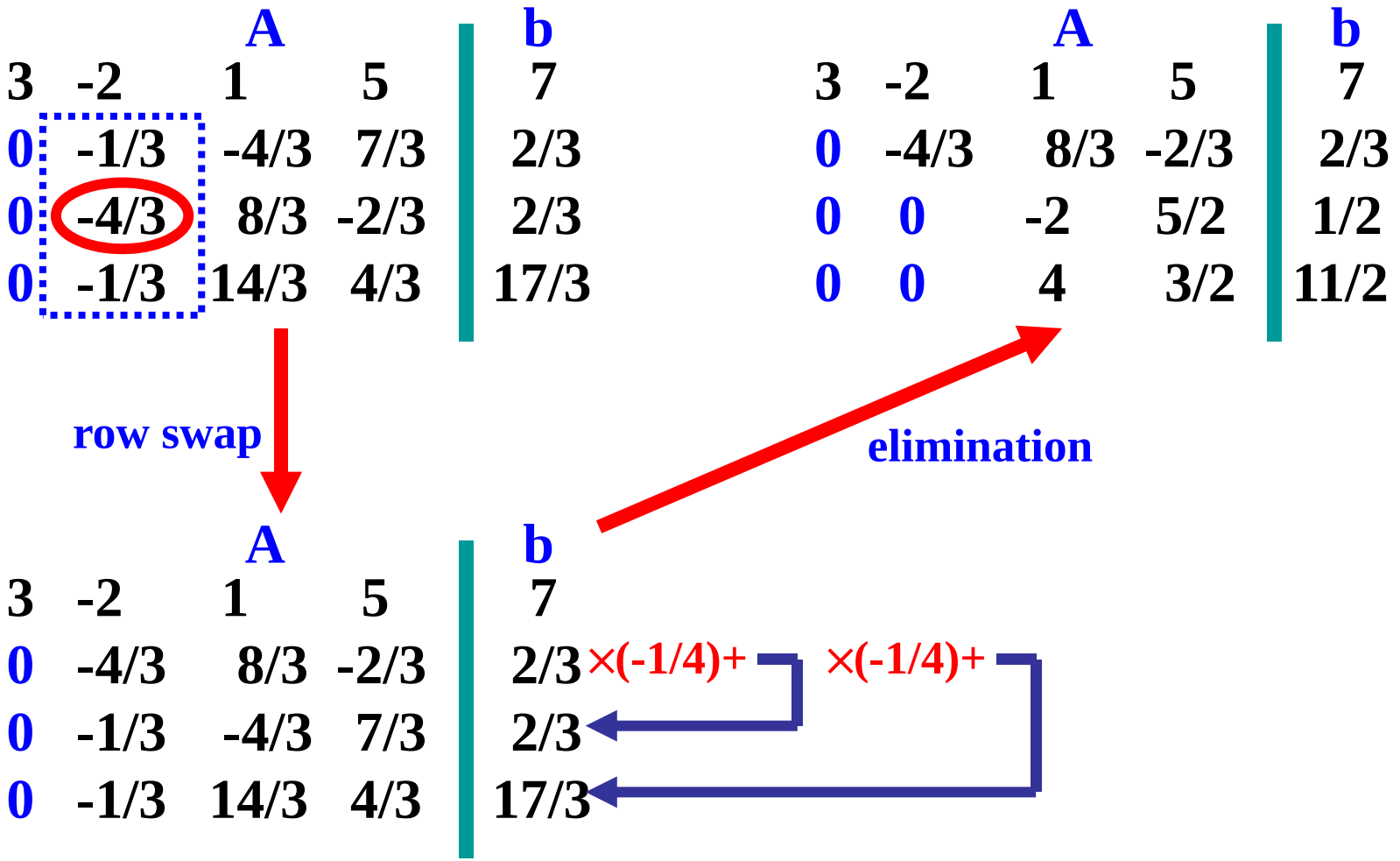
Partial Pivoting: 4/6

- **Blue dashed line: current column, Red circle: max**



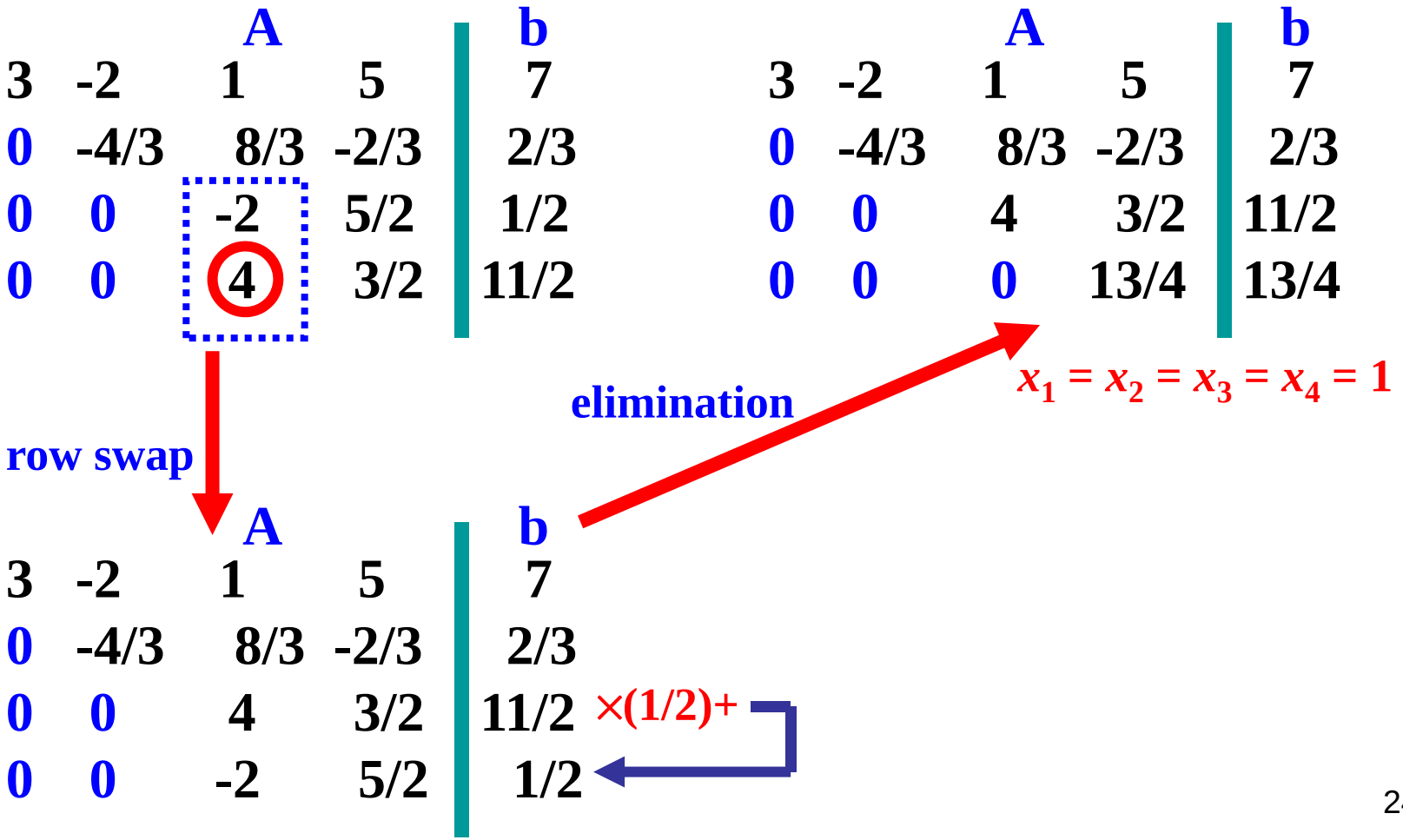
Partial Pivoting: 5/6

- **Blue dashed line: current column, Red circle: max**



Partial Pivoting: 6/6

- **Blue dashed line: current column, Red circle: max**

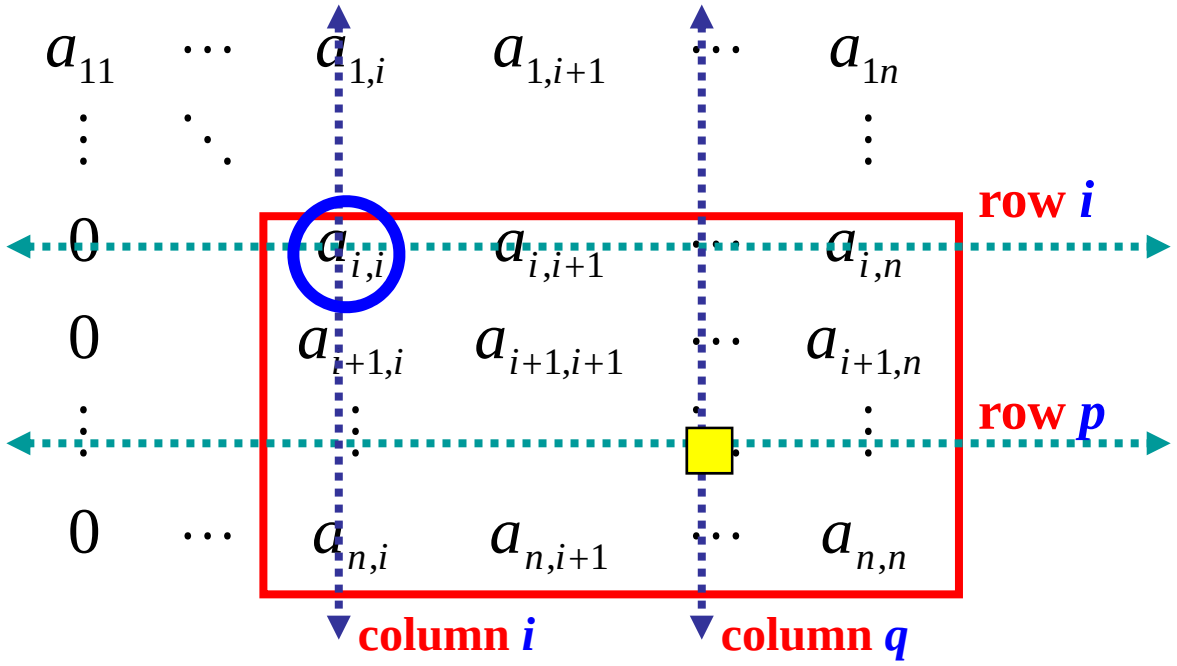


Complete Pivoting: 1/4

- **What if $a_{i,i}$, $a_{i+1,i}$, …, $a_{n,i}$ are all very close to zero when doing pivoting for column i?**
- **If this happens, partial pivoting has a problem, because no matter which row is swapped, there is a chance to have 0/0, overflow or truncation/rounding.**
- **In this case, *complete pivoting* may help.**

Complete Pivoting: 2/4

- With complete pivoting, the block defined by $a_{i,i}$ and $a_{n,n}$ is searched for a maximum $|a_{p,q}|$.
- Then, a row swap (row i and row p) and a column swap (column i and column q) are required.
- After swapping, do the usual elimination.



Complete Pivoting: 3/4

- **While swapping rows does not change the solutions, swapping column *i* and column *q* changes the positions of x_i and x_q.**
- **To overcome this problem, we have to keep track the swapping operations so that column swapping will not affect the solutions.**
- ***One may use an index array*!**

Complete Pivoting: 4/4

- Index array `idx()` is initialized to 1, 2, …, n.
- If columns i and q are swapped, the content of the i-th and q-th entries of `idx()` are also swapped (*i.e.*, swapping `idx(i)` and `idx(q)`).
- At the end, `idx(k) = h` means column k is the solution of x_h. For example, if `idx(1)=4`, `idx(2)=3`, `idx(3)=1` and `idx(4)=2`, this means columns 1, 2, 3 and 4 contain the solutions to x_4, x_3, x_1 and x_2.
- Sometimes, this index array is referred to as a *permutation array*.

Efficiency Concerns

- **Elimination with pivoting does not increase the number of multiplications; however, it does use CPU time for comparisons and swapping.**
- **Although compared with multiplications and divisions swapping may be insignificant, pivoting does add speed penalty to the efficiency of the methods.**
- **One may use index arrays to avoid actually carrying out swapping.**
- **Exercise: *how can this be done?* Refer to the index array method for complete swapping.**

Is Pivoting Really Necessary? 1/3

- **Consider the following without pivoting.**

$$0.0003x + 3.000y = 2.0001$$
$$1.0000x + 1.000y = 1.0000$$

×(-1/0.0003)+

elimination

$$0.0003x + 3.000y = 2.0001$$
$$-9999y = -6666$$

exact arithmetic yields $x = 1/3$ and $y = 2/3$

$$y = 6666 / 9999$$

$$x = \frac{2.0001 - 3.0000 \times (6666 / 9999)}{0.0003}$$

Is Pivoting Really Necessary? 2/3

- **Without pivoting:**

$$y = 6666 / 9999$$

$$x = \frac{2.0001 - 3.0000 \times (6666 / 9999)}{0.0003}$$

Possible cancellation here, as 3×(6666/9999)≈2.0001!

Precision	*y*	*x*
4	0.6667	0
5	0.66667	0.3
6	0.666667	0.33
7	0.6666667	0.333

inaccurate

Is Pivoting Really Necessary? 3/3

- **With pivoting:**

$$1.0000x + 1.0000y = 1.0000 \qquad 1.0000x + 1.0000y = 1.0000$$
$$0.0003x + 3.0000y = 2.0001 \qquad 2.9997y = 1.9998$$

$$y = \frac{1.9998}{2.9997}$$

$$x = 1.0000 - y$$

Precision	*y*	*x*
4	0.6667=2/3	0.3333
5	0.66667	0.33333
6	0.666667	0.333333
7	0.6666667	0.3333333

Pitfalls of Elimination: 1/2

- **The first one is, of course, $a_{i,i} \approx 0$ when computing $-a_{k,i}/a_{i,i}$. This can be overcome with partial or complete pivoting.**
- **However, *singular* systems cannot be solved by elimination (*e.g.*, two parallel lines do not have a solution).**
- **Rounding can be a problem. Even with pivoting, rounding is still there and could propagate from earlier stages to the next.**
- **In general, $n \leq 100$ is OK. Otherwise, consider using other (*e.g.*, indirect) methods.**

Pitfalls of Elimination: 2/2

- ***Ill-Conditioned*** **systems are trouble makers.**
- **Ill-Conditioned systems are systems very sensitive to small changes to coefficients, and there could be many *seemingly correct* "solutions."**
- **Since these "solutions" seem to satisfy the systems, one may be misled to believe they are "correct" solutions.**

LU-Decompositions: 1/8

- **Idea: The basic idea of LU-decomposition is to "decompose" the given matrix $A=[a_{i,j}]$ of $A \bullet x=b$ into a product of a lower triangular and an upper triangular matrices (*i.e.*, $A = L \bullet U$).**
- **The lower triangular matrix has all diagonal elements being 1's (*i.e.*, Doolittle form).**

lower triangular **upper triangular**

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & 1 \end{bmatrix} \bullet \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{n,n} \end{bmatrix}$$

A = L • U

LU-Decompositions: 2/8

- If A has been decomposed as $A=L\cdot U$, then solving $A\cdot x = b$ becomes solving $(L\cdot U)\cdot x = b$.
- $(L\cdot U)\cdot x = b$ can be rewritten as $L\cdot(U\cdot x) = b$.
- Let $y = U\cdot x$. Then, $L\cdot(U\cdot x) = b$ becomes two systems of linear equations: $L\cdot y = b$ and $U\cdot x = y$.
- Since L and b are known, one can solve for y. Once y becomes available, it is used in $U\cdot x = y$ to solve for x.
- This is the key concept of LU-decomposition.

LU-Decompositions: 3/8

- **Would it make sense when one system $A \cdot x = b$ becomes two $L \cdot y = b$ and $U \cdot x = y$?**
- **It depends; however, both systems are very easy to solve if $A = L \cdot U$ is available.**
- **Can use backward substitution to solve $U \cdot x = y$.**

$$\begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,i} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \cdots & u_{2,i} & \cdots & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & u_{i,i} & \cdots & u_{i,n} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & u_{n,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix}$$

LU-Decompositions: 4/8

- **The $\mathbf{L \cdot y = b}$ system is also easy to solve.**
- **From row 1, we have $y_1 = b_1$. Row 2 is $l_{2,1}y_1 + y_2 = b_2$. Row i is**

$$l_{i,1}y_1 + l_{i,2}y_2 + \cdots + l_{i,i-1}y_{i-1} + y_i = b_i$$

- **Hence,** $y_i = b_i - \left(l_{i,1}y_1 + l_{i,2}y_2 + \cdots + l_{i,i-1}y_{i-1}\right) = b_i - \sum_{k=1}^{i-1} l_{i,k}y_k$

$$\begin{matrix} \\ \\ \\ \textbf{row } i \\ \\ \\ \end{matrix} \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ l_{i,1} & l_{i,2} & \cdots & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & l_{n,i} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

LU-Decompositions: 5/8

- **The following code is based on the formula mentioned earlier**

$$y_i = b_i - \left(l_{i,1} y_1 + l_{i,2} y_2 + \cdots + l_{i,i-1} y_{i-1} \right) = b_i - \sum_{k=1}^{i-1} l_{i,k} y_k$$

- **This is referred to as forward substitution since $y_1, y_2, \ldots, y_{i-1}$ are used to compute y_i.**

```
y1 = b1
DO i = 2, n
   yi = bi
   DO k = 1, i-1
      yi = yi - Li,k*yk
   END DO
END DO
```

This is an $O(n^2)$ method
Do it yourself.

LU-Decompositions: 6/8

- **In summary, LU-decomposition methods have the following procedure:**
  - **From A in $A \cdot x = b$ find a LU-decomposition $A = L \cdot U$**
  - **Solve for y with forward substitution from $L \cdot y = b$**
  - **Solve for x with backward substitution from $U \cdot x = y$**

LU-Decompositions: 7/8

- Why LU-decomposition when Gaussian elimination is available?
- **The reason is simple: saving time.**
- **Suppose we need to solve *k* systems of linear equations like this: $A \cdot x_1 = b_1$, $A \cdot x_2 = b_2$, $A \cdot x_3 = b_3$, …, $A \cdot x_k = b_k$. Note that they share the same A and not all b_i's are available at the same time.**
- **Without a LU-decomposition, Gaussian elimination would be repeated *k* times, one for each system. This is time consuming.**

LU-Decompositions: 8/8

- **Since each Gaussian elimination requires $O(n^3)$ multiplications, solving k systems requires $O(k \times n^3)$ multiplications.**
- **A LU-decomposition decomposes $A = L \cdot U$.**
- **For each b_i, applying a forward followed by a backward substitution yields the solution x_i.**
- **Since each backward and forward substitution requires $O(n^2)$ multiplications, solving k systems requires $O(n^3 + k \times n^2)$ multiplications.**
- **Therefore, LU-decomposition is faster!**

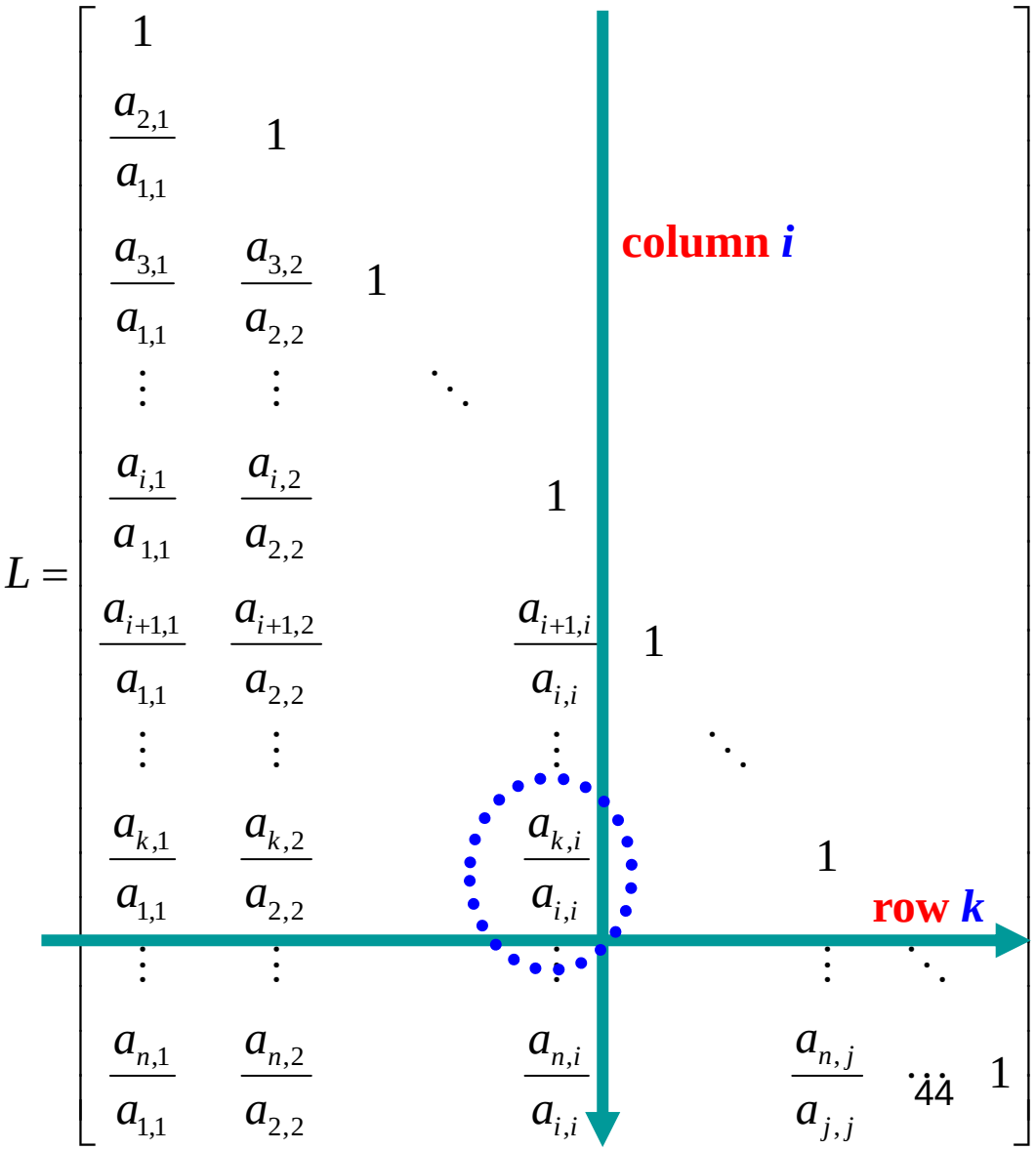
elimination is still needed

How to Decompose: 1/4

- **LU-decomposition is easier than you thought!**
- ***Gaussian elimination generates an upper triangular matrix, which is the* U *in* A = L·U.**
- **More importantly, the *elimination process also produces lower triangular matrix* L, although we never thought about this possibility.**
- **The next few slides shows how to recover this lower triangular matrix L during the elimination process.**

How to Decompose: 2/4

- When handling row i, $-a_{k,i}/a_{i,i}$ is multiplied to row i and the result is added to row k.
- The entry on row k and column i of L, where $k > i$, is $a_{k,j}/a_{j,j}$.
- Thus, L can be generated on-the-fly as $a_{k,i} / a_{i,i}$ is a multiplier!



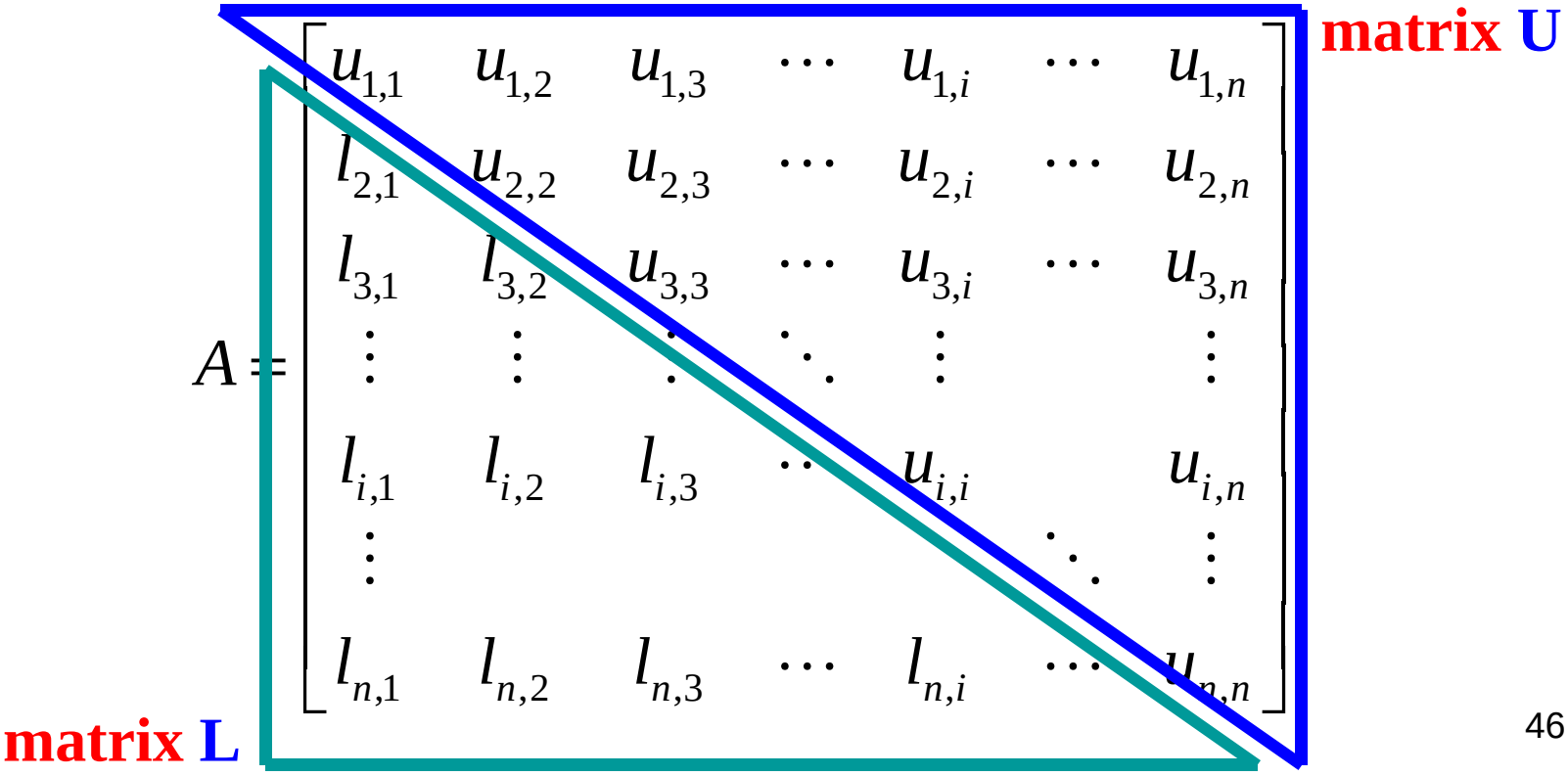
How to Decompose: 3/4

- During Gaussian elimination, the lower triangular part is set to zero.
- One may use this portion for the lower triangular matrix L without its diagonal; however, the diagonal is part of U rather than L.

```
DO i = 1, n-1                          ! using row i, i.e., a(i,i)
  DO k = i+1, n                        ! we want to eliminate a(k,i)
    S = a(k,i)/a(i,i)
    DO j = i+1, n                      ! for each entry on row k
      a(k,j) = a(k,j) - S*a(i,j)       !   update its value
    END DO
    a(k,i) = S                         ! save this "multiplier"
  END DO                               ! don't forget to update b(k)
END DO
```

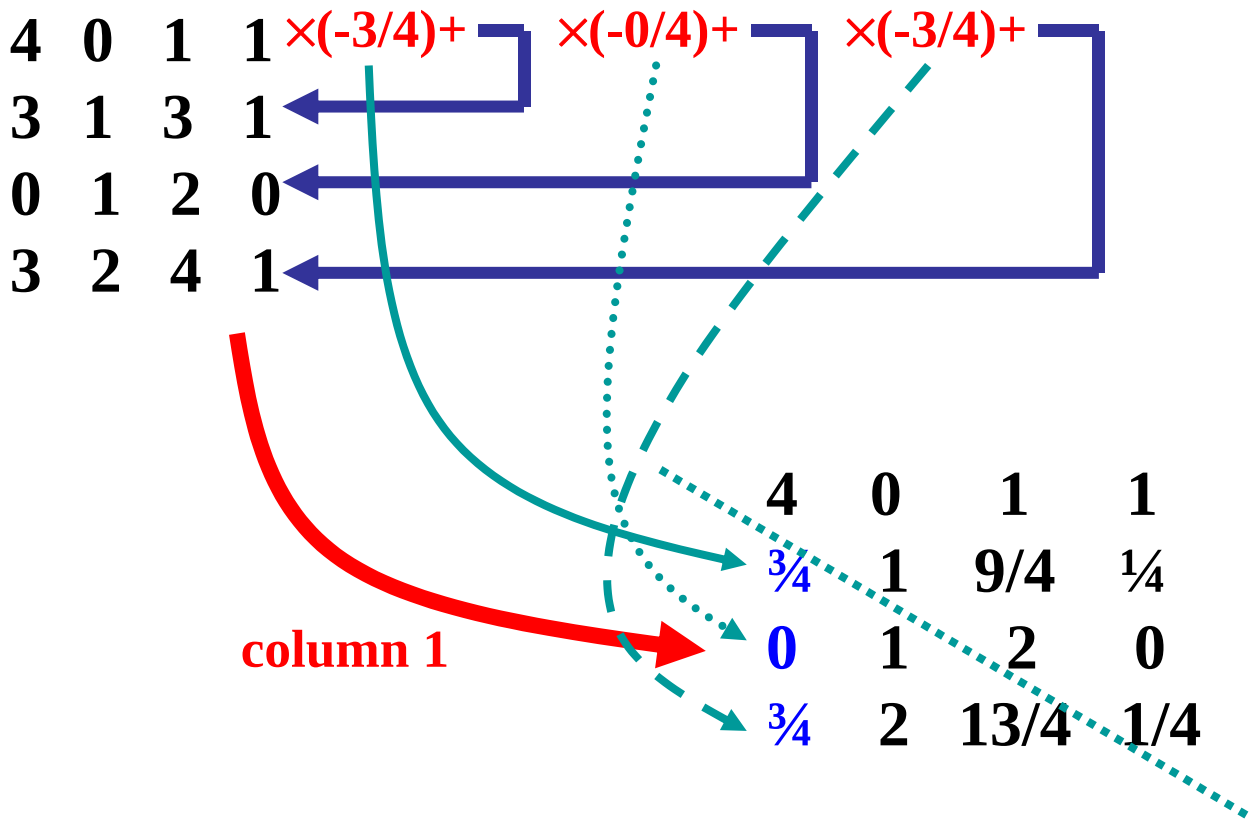
How to Decompose: 4/4

- **After the decomposition A = L·U, matrix A is replaced by U in the upper triangular part and L in the lower triangular part *without the diagonal*.**



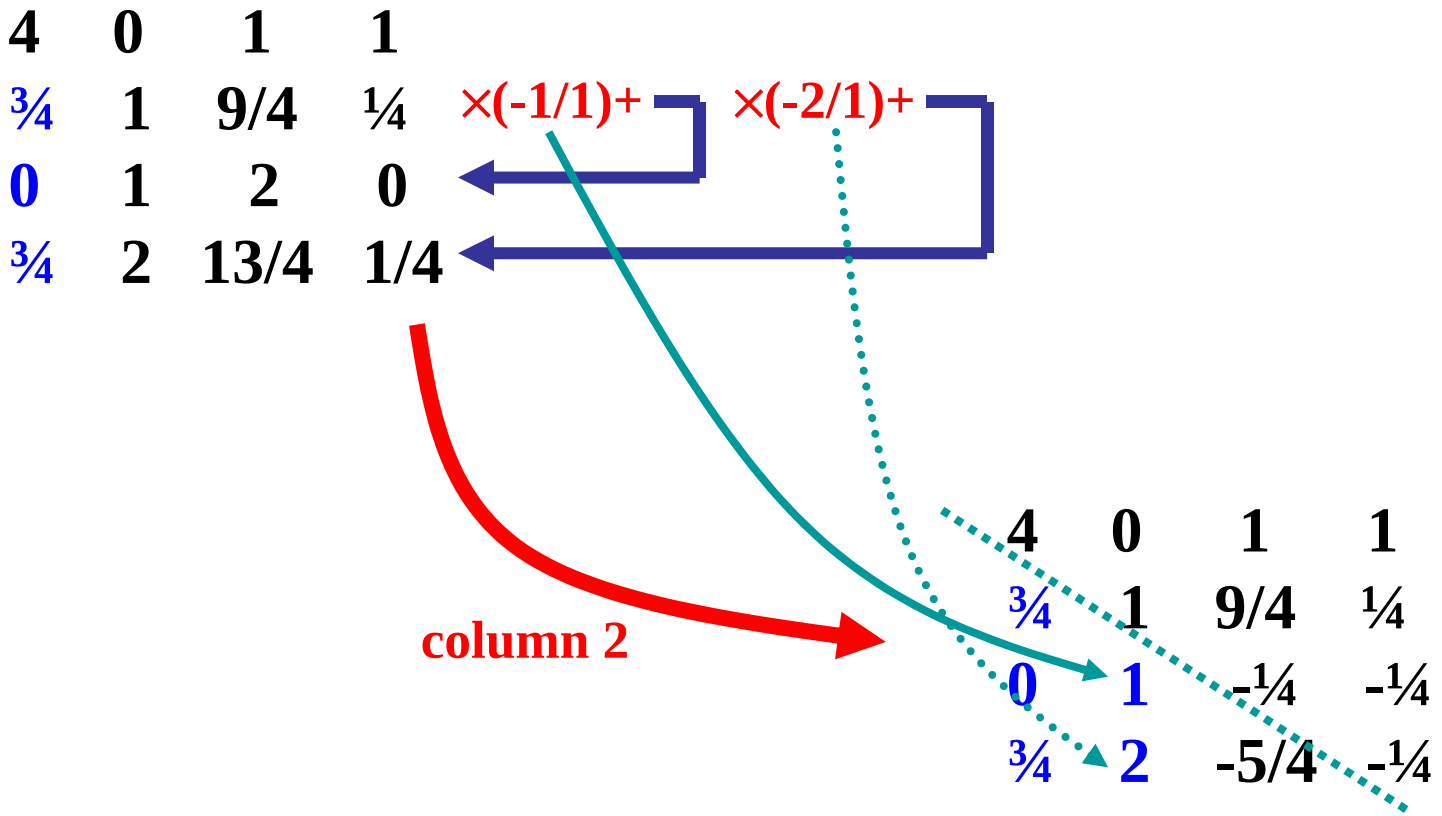
Example: 1/4

- **Blue: values of the lower diagonal portion (*i.e.*, matrix L without the diagonal)**



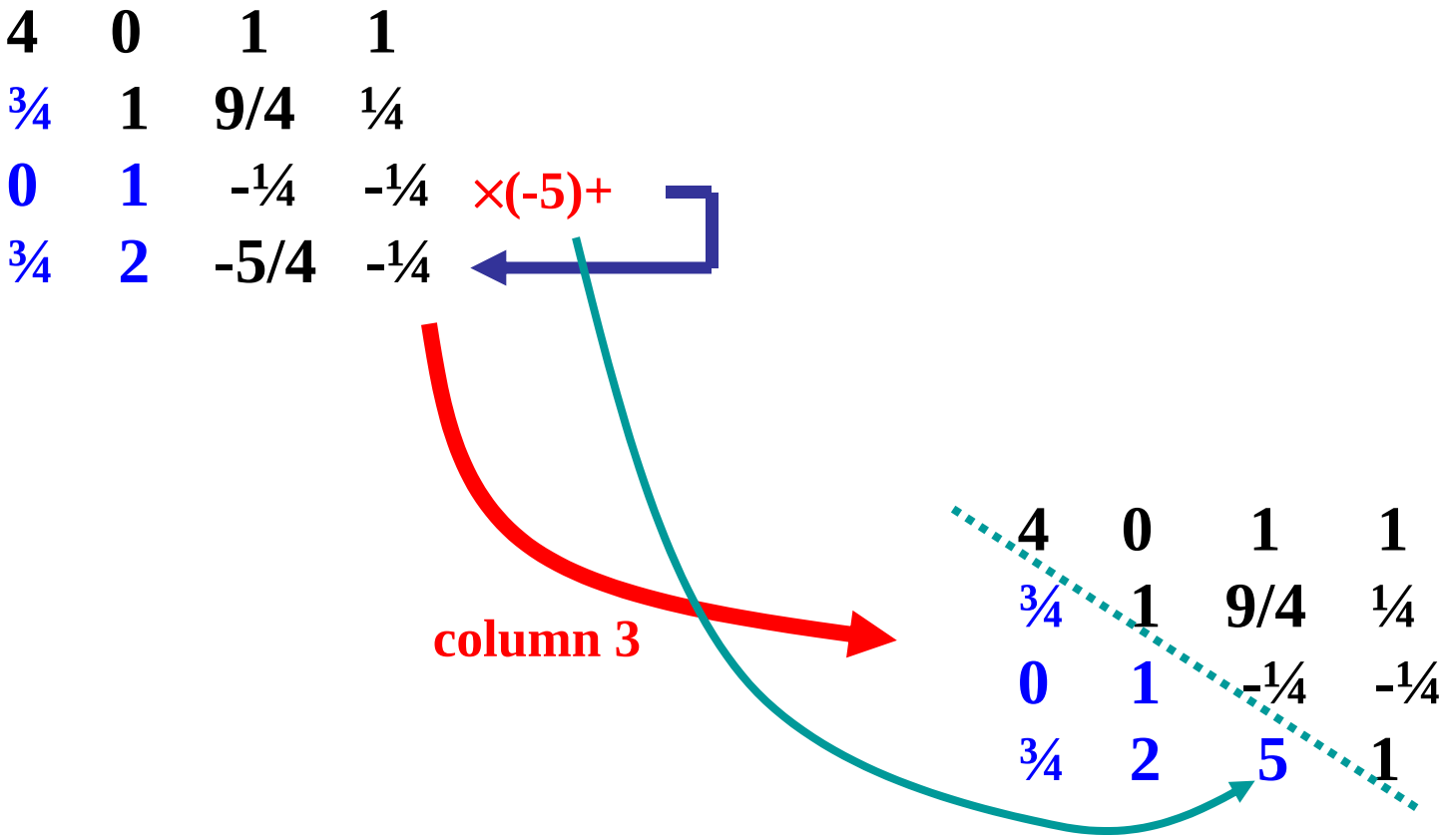
Example: 2/4

- **Blue: values of the lower diagonal portion (*i.e.*, matrix L without the diagonal)**



Example: 3/4

- **Blue: values of the lower diagonal portion (*i.e.*, matrix L without the diagonal)**



Example: 4/4

- **Verification:**

$$\begin{bmatrix} 1 & & & \\ 3/4 & 1 & & \\ 0 & 1 & 1 & \\ 3/4 & 2 & 5 & 1 \end{bmatrix} \cdot \begin{bmatrix} 4 & 0 & 1 & 1 \\ & 1 & 9/4 & 1/4 \\ & & -1/4 & -1/4 \\ & & & 1 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 1 & 1 \\ 3 & 1 & 3 & 1 \\ 0 & 1 & 2 & 0 \\ 3 & 2 & 4 & 1 \end{bmatrix}$$

L • U = A

How to Solve: 1/2

- **The following is a forward substitution solving for y from L and b (*i.e.*, L·y = b):**

```
y1 = b1
DO i = 2, n
   yi = bi
   DO k = 1, i-1
      yi = yi - ai,k*yk
   END DO
END DO
```

$$A = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,i} & \cdots & u_{1,n} \\ l_{2,1} & u_{2,2} & u_{2,3} & \cdots & u_{2,i} & \cdots & u_{2,n} \\ l_{3,1} & l_{3,2} & u_{3,3} & \cdots & u_{3,i} & \cdots & u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots \\ l_{i,1} & l_{i,2} & l_{i,3} & \cdots & u_{i,i} & & u_{i,n} \\ \vdots & & & & & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \cdots & l_{n,i} & \cdots & u_{n,n} \end{bmatrix}$$

$a_{i,k}$ **is actually** $l_{i,k}$

How to Solve: 2/2

- **The following is a backward substitution solving for x from U·x = y.**

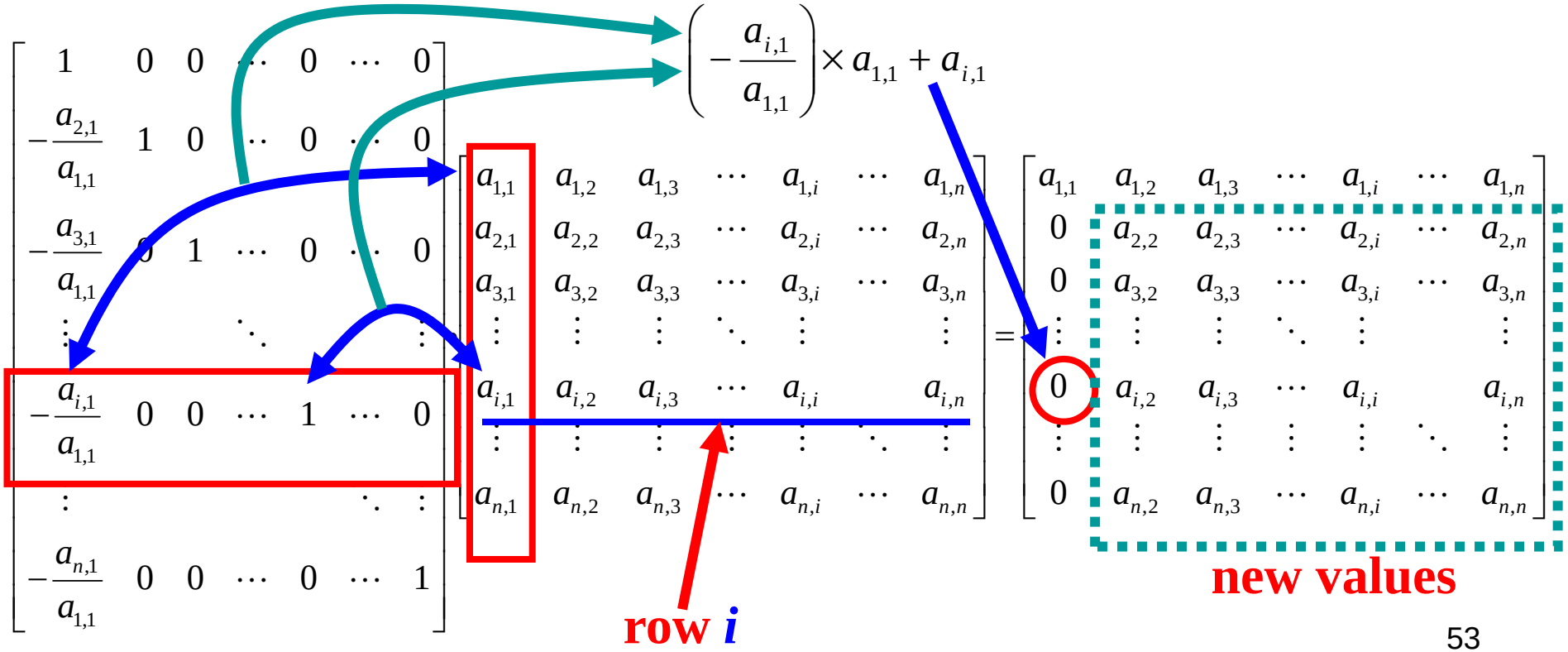
```
DO i = n, 1, -1
   S = y_i
   DO k = i+1,n
      S = S - a_i,k*x_k
   END DO
   x_i = S/a_i,i
END DO
```

$$A = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,i} & \cdots & u_{1,n} \\ l_{2,1} & u_{2,2} & u_{2,3} & \cdots & u_{2,i} & \cdots & u_{2,n} \\ l_{3,1} & l_{3,2} & u_{3,3} & \cdots & u_{3,i} & \cdots & u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots \\ l_{i,1} & l_{i,2} & l_{i,3} & \cdots & u_{i,i} & & u_{i,n} \\ \vdots & & & & & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \cdots & l_{n,i} & \cdots & u_{n,n} \end{bmatrix}$$

$a_{i,k}$ here is $u_{i,k}$!

Why Is L Correct? 1/10

- **For row 1, the elimination step multiplies every element on row 1 by $-a_{i,1}/a_{1,1}$ and adds the results to row *i*. *This is a matrix multiplication!***



Why Is L Correct? 2/10

- **Define matrix E_1 as follows. Then, $E_1 \cdot A$ sets all entries on column 1 below $a_{1,1}$ to zero.**

$$E_1 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ -a_{2,1}/a_{1,1} & 1 & 0 & \cdots & 0 & \cdots & 0 \\ -a_{3,1}/a_{1,1} & 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & & & \ddots & & & \vdots \\ -a_{i,1}/a_{1,1} & 0 & 0 & \cdots & 1 & \cdots & 0 \\ \vdots & & & & & \ddots & \vdots \\ -a_{n,1}/a_{n,n} & 0 & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

E_1 is lower triangular!

Why Is L Correct? 3/10

- **Define E_2 as follows. The same calculation shows $E_2 \cdot (E_1 \cdot A)$ sets all entries of $E_1 \cdot A$ below $a_{2,2}$ to 0.**

$$E_2 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & -a_{3,2}/a_{2,2} & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & & \ddots & & & \vdots \\ 0 & -a_{i,2}/a_{2,2} & 0 & \cdots & 1 & \cdots & 0 \\ \vdots & \vdots & & & & \ddots & \vdots \\ 0 & -a_{n,2}/a_{2,2} & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

E_2 is lower triangular!

Why Is L Correct? 4/10

- **For column *i*, lower triangular matrix E_i is defined as follows. It has all the "multipliers".**

$$E_i = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \ddots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & & -\frac{a_{i+1,i}}{a_{i,i}} & \ddots & & & \vdots \\ 0 & & -\frac{a_{i+2,i}}{a_{i,i}} & \cdots & 1 & \cdots & 0 \\ \vdots & & \vdots & & & \ddots & \vdots \\ 0 & & -\frac{a_{n,i}}{a_{i,i}} & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

Why Is L Correct? 5/10

- **Since $E_{i-1}\cdot E_{i-2}\cdot\ldots\cdot E_2\cdot E_1\cdot A$ sets the lower diagonal part of column 1 to column i-1 to 0, the new matrix $E_i\cdot(E_{i-1}\cdot E_{i-2}\cdot\ldots\cdot E_2\cdot E_1\cdot A)$ eliminates the lower diagonal part of $a_{i,i}$.**

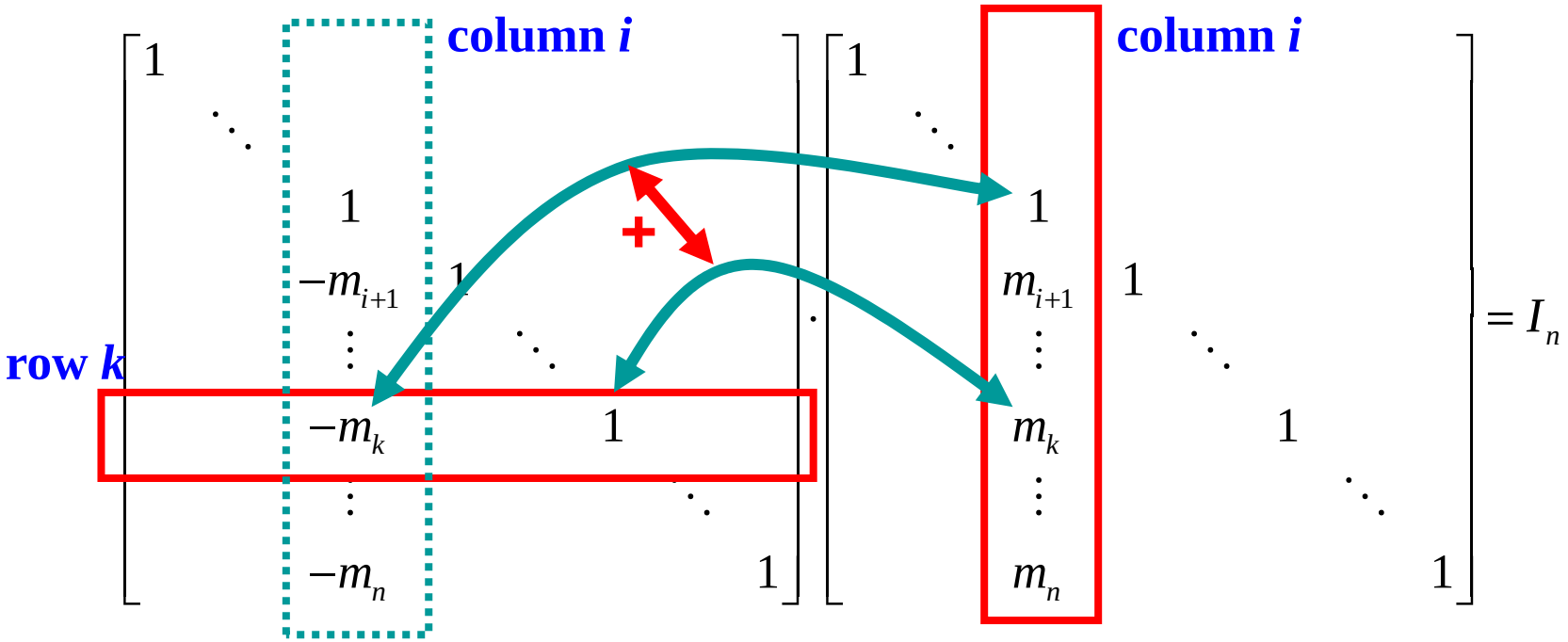
$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \ddots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & & -\frac{a_{i+1,i}}{a_{i,i}} & \ddots & & & \vdots \\ 0 & & -\frac{a_{i+2,i}}{a_{i,i}} & \cdots & 1 & \cdots & 0 \\ \vdots & & \vdots & & & \ddots & \vdots \\ 0 & & -\frac{a_{n,i}}{a_{i,i}} & \cdots & 0 & \cdots & 1 \end{bmatrix}}_{E_i} \cdot \underbrace{\begin{bmatrix} a_{1,1} & \cdots & a_{1,i} & a_{1,i+1} & a_{1,i+2} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & \cdots & a_{i,i} & a_{i,i+1} & a_{i,i+2} & \cdots & a_{i,n} \\ \vdots & & a_{i+1,i} & a_{i+1,i+1} & a_{i+1,i+2} & & a_{i+1,n} \\ 0 & & a_{i+2,i} & a_{i+2,i+1} & a_{i+2,i+2} & & a_{i+2,n} \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{n,i} & a_{n,i+1} & a_{n,i+2} & \cdots & a_{n,n} \end{bmatrix}}_{E_{i-1}\cdot E_{i-2}\cdot\ldots\cdot E_2\cdot E_1\cdot A} = \underbrace{\begin{bmatrix} a_{1,1} & \cdots & a_{1,i} & a_{1,i+1} & a_{1,i+2} & \cdots & a_{1,n} \\ 0 & \ddots & \vdots & \vdots & \vdots & \cdots & a_{2,n} \\ 0 & \cdots & a_{i,i} & a_{i,i+1} & a_{i,i+2} & \cdots & a_{i,n} \\ \vdots & & 0 & a_{i+1,i+1} & a_{i+1,i+2} & & a_{i+1,n} \\ 0 & & 0 & a_{i+2,i+1} & a_{i+2,i+2} & & a_{i+2,n} \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n,i+1} & a_{n,i+2} & \cdots & a_{n,n} \end{bmatrix}}_{E_i\cdot E_{i-1}\cdot E_{i-2}\cdot\ldots\cdot E_2\cdot E_1\cdot A}$$

Why Is L Correct? 6/10

- **Repeating this process, matrices E_1, E_2, …, E_{n-1} are constructed so that $E_{n-1} \cdot E_{n-2} \cdot \ldots \cdot E_2 \cdot E_1 \cdot A$ is upper diagonal.**
- **Note that only E_1, E_2, …, E_{n-1} are needed, because the lower diagonal part has $n-1$ columns.**
- **Therefore, *the elimination process is completely described by the product of matrices* E_i's.**
- **Now, we have $E_{n-1} \cdot E_{n-2} \cdot \ldots \cdot E_2 \cdot E_1 \cdot A = U$, where U is an upper triangular matrix, and $A = (E_{n-1} \cdot E_{n-2} \cdot \ldots \cdot E_2 \cdot E_1)^{-1} \cdot U$, where T^{-1} means the inverse matrix of T (*i.e.*, $T \cdot T^{-1} = I$, the identify matrix).**
- **What is $(E_{n-1} \cdot E_{n-2} \cdot \ldots \cdot E_2 \cdot E_1)^{-1}$? In fact, this is matrix L we want!**

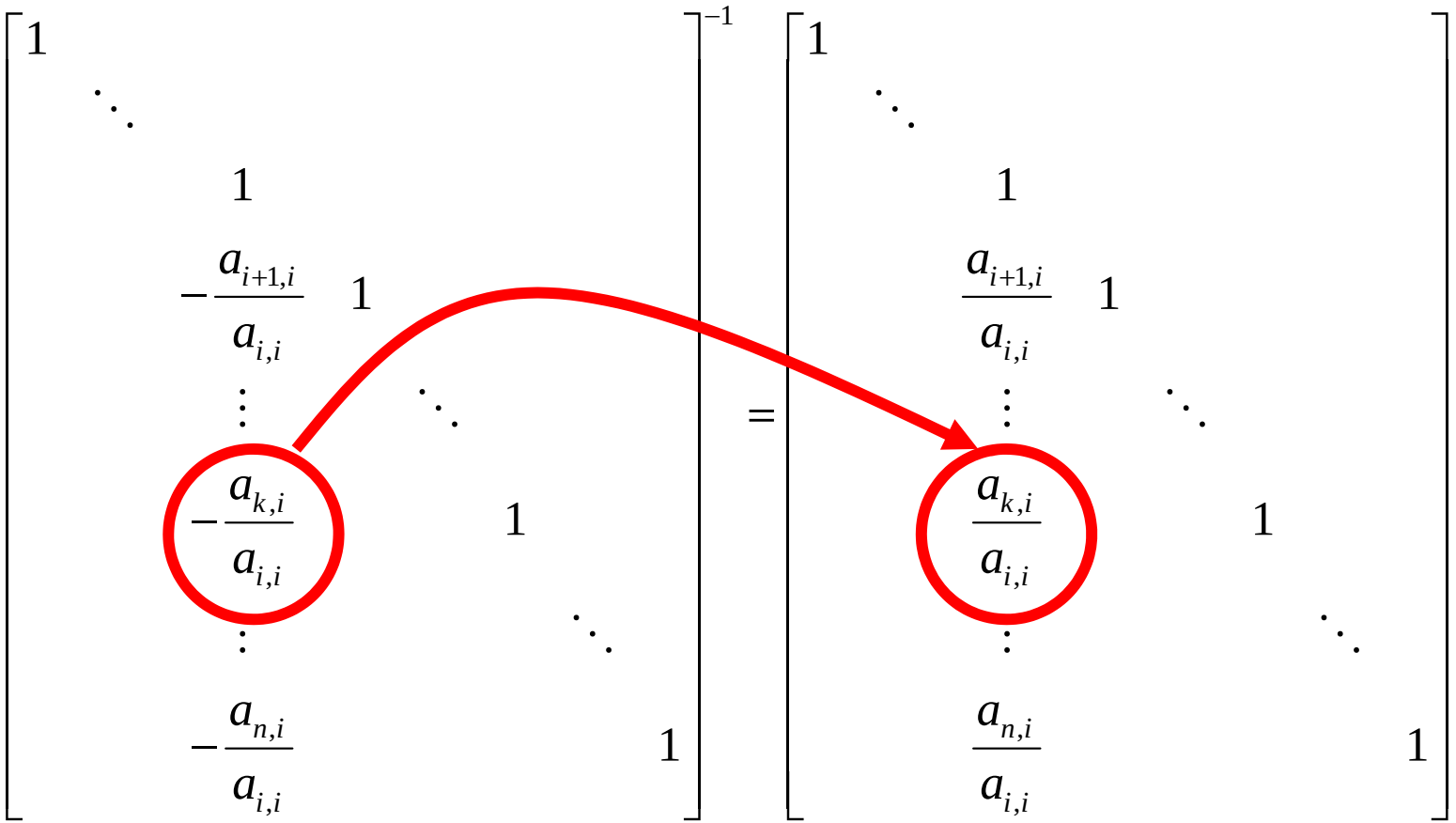
Why Is L Correct? 7/10

- **In linear algebra, the inverse of $A \cdot B$, $(A \cdot B)^{-1}$, is computed as $(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$.**
- **$(E_{n-1} \cdot E_{n-2} \cdot \ldots \cdot E_2 \cdot E_1)^{-1} = E_1^{-1} \cdot E_2^{-1} \cdot \ldots \cdot E_{n-2}^{-1} \cdot E_{n-1}^{-1}$.**
- **E_i^{-1} is very easy to compute! See below.**



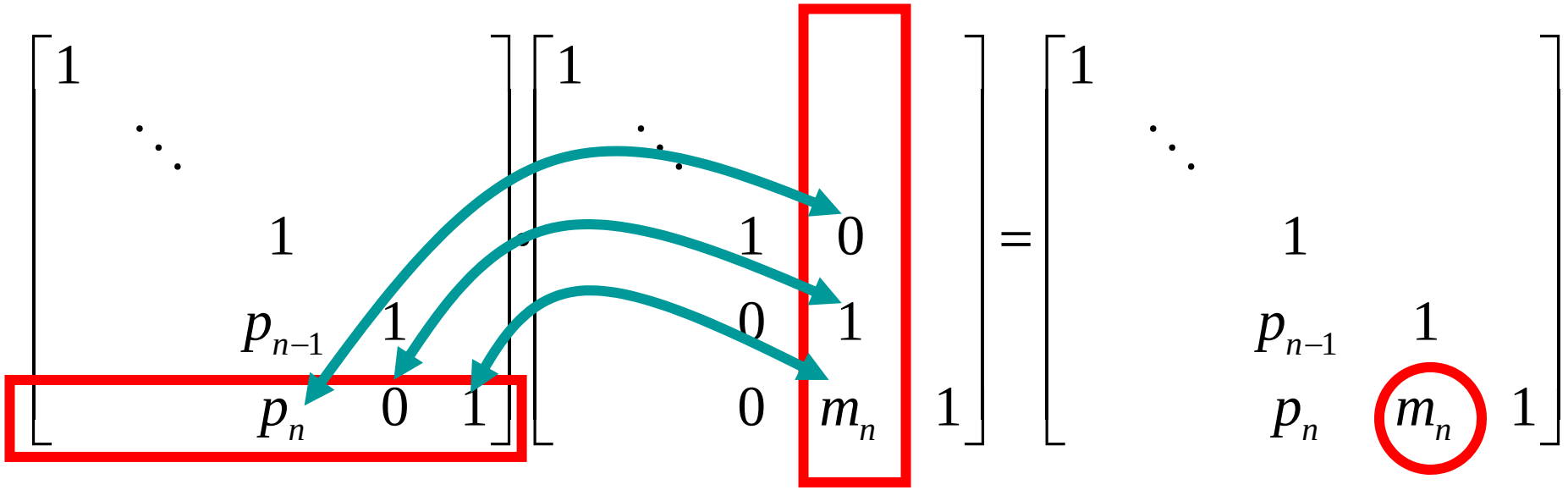
Why Is L Correct? 8/10

- **Therefore, E_i^{-1} is obtained by removing the negative signs from E_i!**



Why Is L Correct? 9/10

- **Additionally, $E_1^{-1} \cdot E_2^{-1} \cdot \ldots \cdot E_{n-2}^{-1} \cdot E_{n-1}^{-1}$ is also easy to compute.**
- **The following shows the results of $E_{n-2}^{-1} \cdot E_{n-1}^{-1}$. It is basically "filling" the entries.**



Why Is L Correct? 10/10

- **$E_1^{-1} \cdot E_2^{-1} \cdot \ldots \cdot E_{n-2}^{-1} \cdot E_{n-1}^{-1}$ is computed by removing the signs from all E_i's and collect them all together into a lower triangular matrix.**
- **Verify this yourself!**

$$E_1^{-1} \bullet E_2^{-1} \bullet \cdots \bullet E_{n-1}^{-1} = \begin{bmatrix} 1 & & & & & & \\ \frac{a_{2,1}}{a_{1,1}} & 1 & & & & & \\ \frac{a_{3,1}}{a_{1,1}} & \frac{a_{3,2}}{a_{2,2}} & 1 & & & & \\ \vdots & \vdots & \ddots & & & & \\ \frac{a_{i,1}}{a_{1,1}} & \frac{a_{i,2}}{a_{2,2}} & & 1 & & & \\ \frac{a_{i+1,1}}{a_{1,1}} & \frac{a_{i+1,2}}{a_{2,2}} & & \frac{a_{i+1,i}}{a_{i,i}} & 1 & & \\ \vdots & \vdots & & \vdots & \ddots & & \\ \frac{a_{k,1}}{a_{1,1}} & \frac{a_{k,2}}{a_{2,2}} & & \frac{a_{k,i}}{a_{i,i}} & & 1 & \\ \vdots & \vdots & & \vdots & & \vdots & \ddots \\ \frac{a_{n,1}}{a_{1,1}} & \frac{a_{n,2}}{a_{2,2}} & & \frac{a_{n,i}}{a_{i,i}} & & \frac{a_{n,j}}{a_{j,j}} & \cdots & 1 \end{bmatrix}$$

Note that the $a_{i,j}$'s are not the original of matrix A. They are computed intermediate results in the elimination process

Pivoting in LU-Decomposition: 1/7

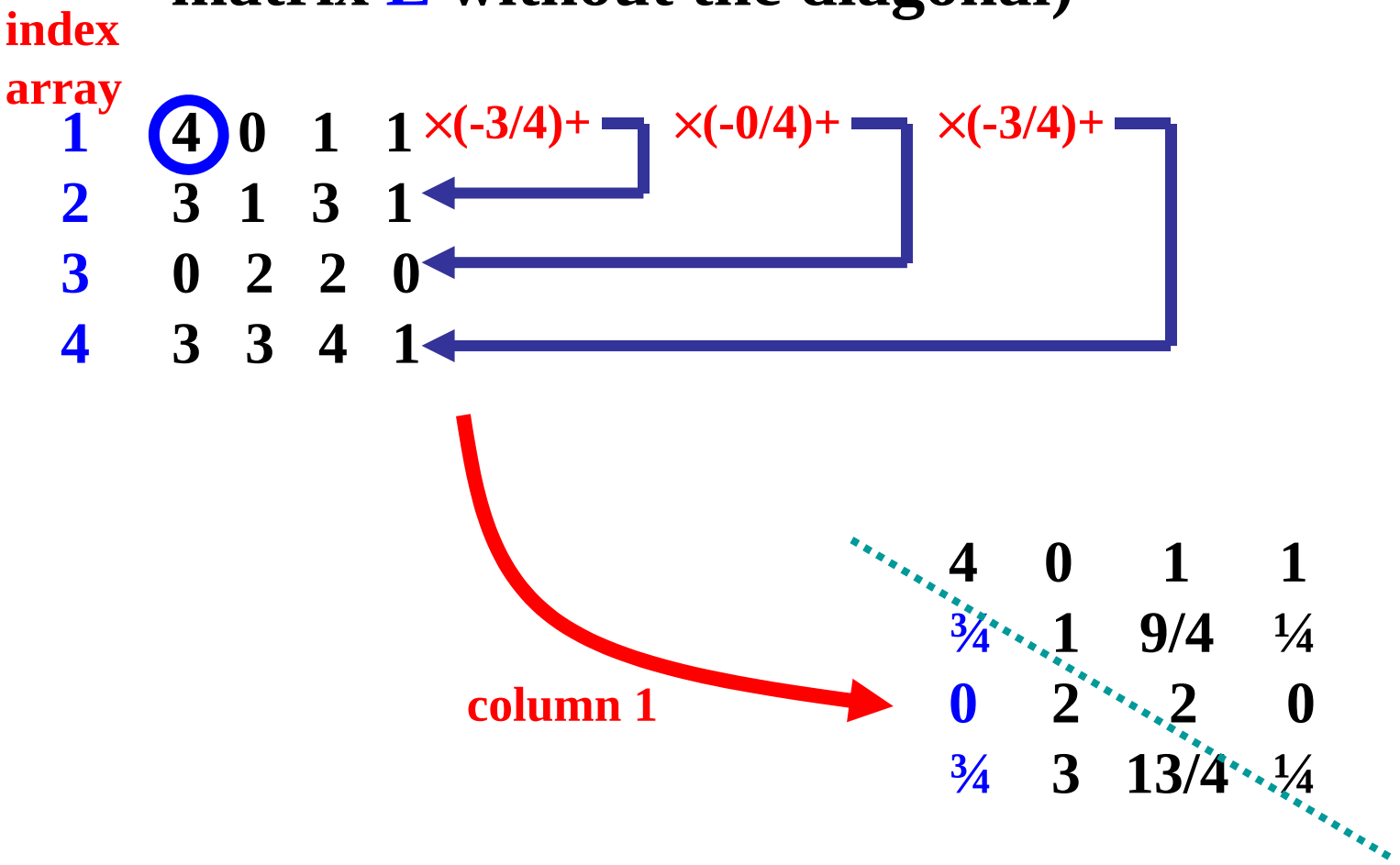
- **LU-Decomposition may also use pivoting.**
- **Although partial pivoting in Gaussian elimination does not affect the order of the solutions, it does in LU-decomposition because LU-decomposition only processes matrix A!**
- **Therefore, when using partial pivoting with LU-decomposition, an index array is needed to save row swapping activities so that the same row swapping can also apply to matrix b later.**

Pivoting in LU-Decomposition: 2/7

- **To keep track *partial pivoting*, an index array `idx()` is needed, which will be used to swap elements of matrix `b`.**
- **For example, if `idx(1)=4`, `idx(2)=1`, `idx(3)=3` and `idx(4)=2`, this means after row swapping, row 1 is equation 4, row 2 is equation 1, row 3 is equation 3, and row 4 is equation 2.**
- **Before using forward/backward substitution, we need to swap b_4 to the first position, b_1, b_3 and b_2 to the 2nd, 3rd and 4th, respectively.**

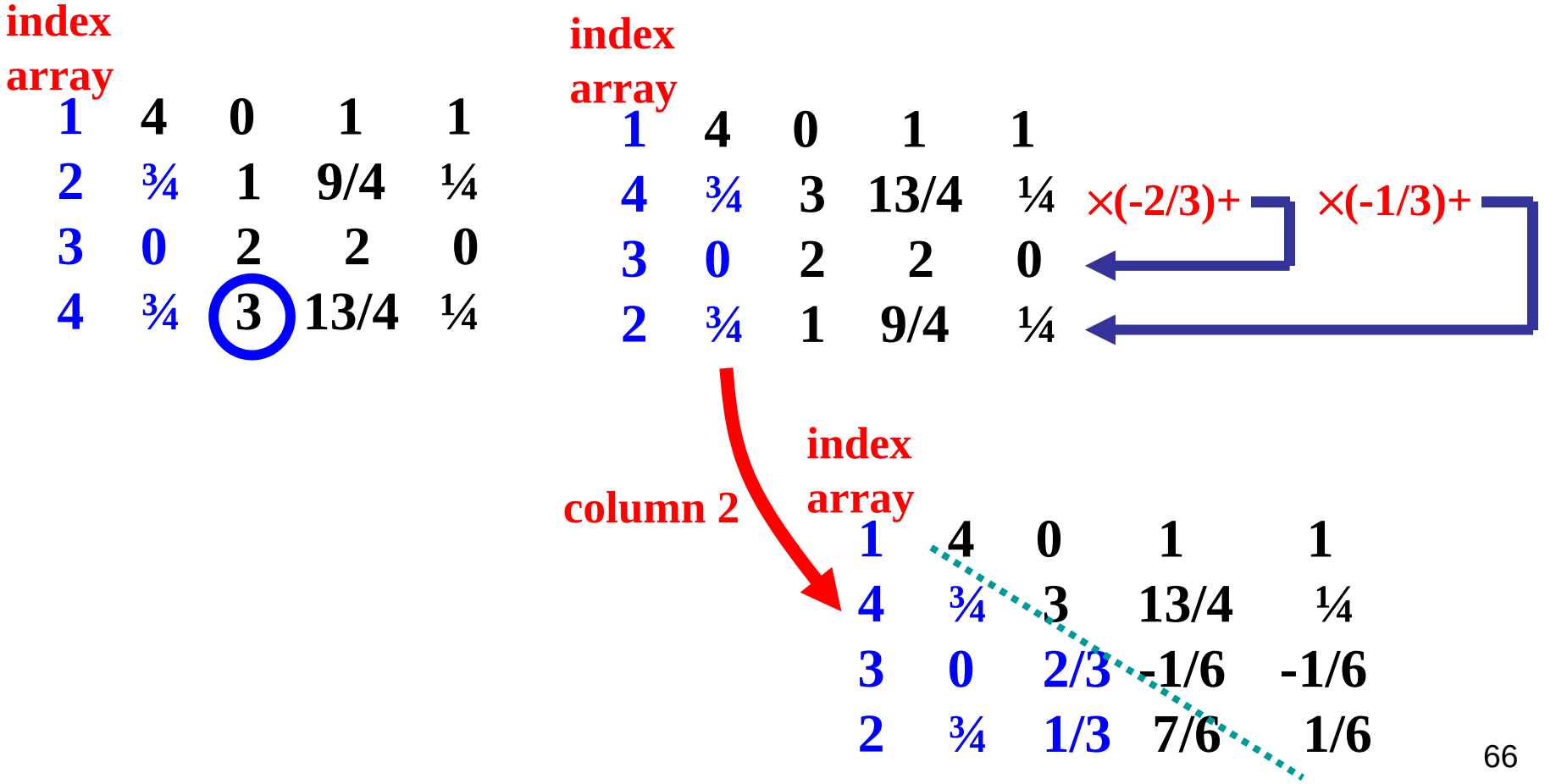
Pivoting in LU-Decomposition: 3/7

- **Blue: values of the lower diagonal portion (*i.e.*, matrix L without the diagonal)**



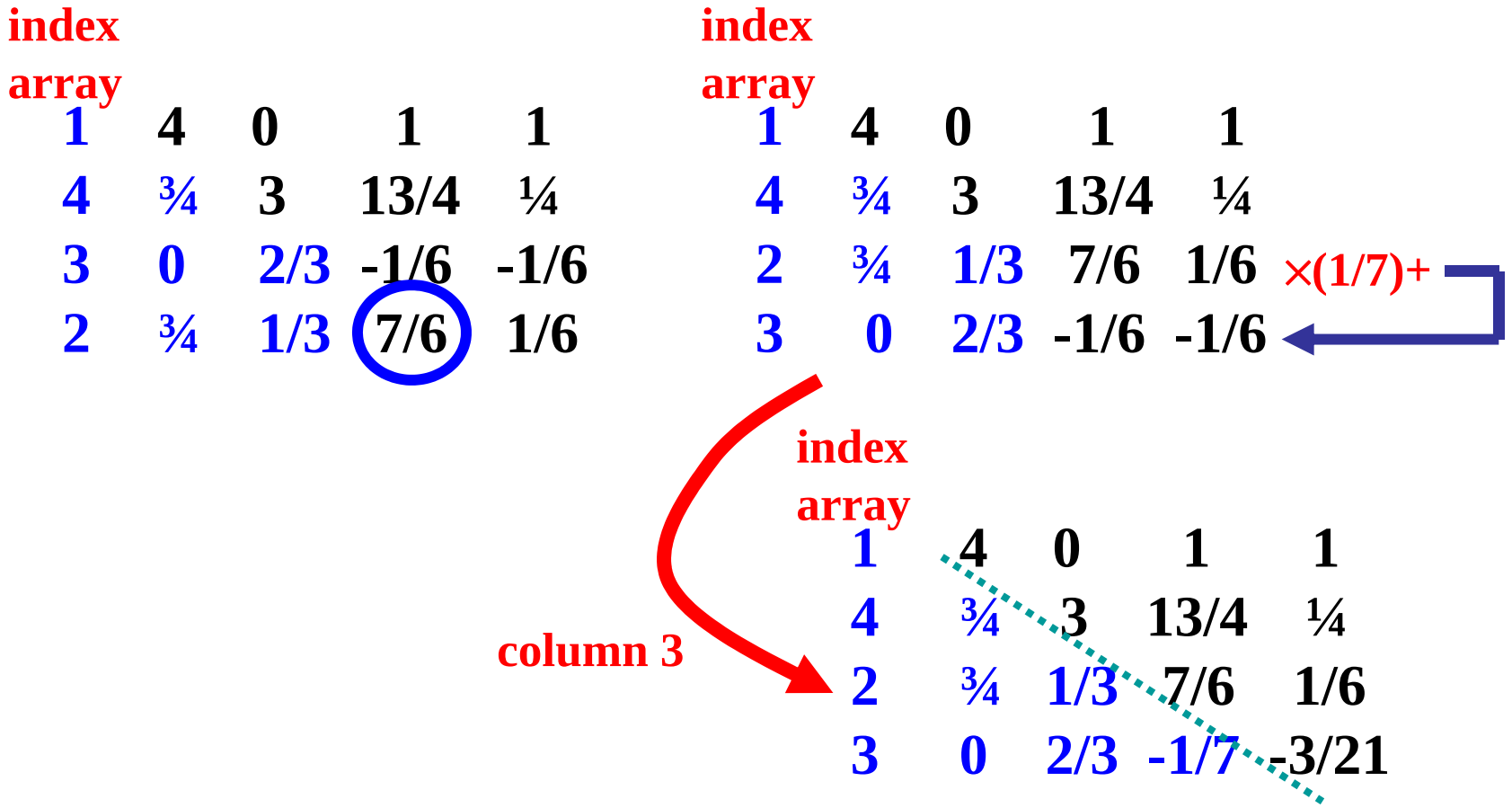
Pivoting in LU-Decomposition: 4/7

- **Blue: values of the lower diagonal portion (*i.e.*, matrix L without the diagonal)**



Pivoting in LU-Decomposition: 5/7

- **Blue: values of the lower diagonal portion (*i.e.*, matrix L without the diagonal)**



Pivoting in LU-Decomposition: 6/7

- **Verification:**

$$\begin{bmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{3}{4} & \frac{1}{3} & 1 & \\ 0 & \frac{2}{3} & \frac{-1}{7} & 1 \end{bmatrix} \cdot \begin{bmatrix} 4 & 0 & 1 & 1 \\ & 3 & \frac{13}{4} & \frac{1}{4} \\ & & \frac{7}{6} & \frac{1}{6} \\ & & & \frac{-3}{21} \end{bmatrix} = \begin{bmatrix} 4 & 0 & 1 & 1 \\ 3 & 3 & 4 & 1 \\ 3 & 1 & 3 & 1 \\ 0 & 2 & 2 & 0 \end{bmatrix} \begin{matrix} \textbf{index} \\ \textbf{array} \\ \mathbf{1} \\ \mathbf{4} \\ \mathbf{2} \\ \mathbf{3} \end{matrix}$$

L • U = A

Pivoting in LU-Decomposition: 7/7

- **LU-decomposition may use complete pivoting.**
- **A second index array is needed to keep track the swapping of variables.**
- **After the decomposition $A = L \cdot U$ is found, the partial pivoting index array is used to swap matrix b's elements, then use the new b to solve for the x.**
- **If complete pivoting is used, the second array is used to get the correct order of variables back.**
- **See Complete Pivoting slides for the details.**

Iterative Methods: 1/2

- **In addition to Gaussian elimination, LU-decomposition, and other methods that guarantee to compute the solution in a fixed number of steps, there are *iterative methods*.**
- **Like we learned in solving non-linear equations, iterative methods start with an initial guess $\mathbf{x}_0$ of $\mathbf{A \bullet x = b}$ and successively compute $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ until the error vector $\mathbf{e}_k = \mathbf{A \bullet x}_k \mathbf{- b}$ is small enough.**
- **In many cases (*e.g.*, very large systems), iterative methods may be more effective than the non-iterative methods.**

Iterative Methods: 2/2

- **There are many iterative methods, from the very classical ones (*e.g.*, Jacobi and Gauss-Seidel) to very modern ones (*e.g.*, conjugate gradient).**
- **We only discuss the classical ones, Jacobi and Gauss-Seidel.**
- **Typical iterative methods have a general form similar to the following:**

$$x_{k+1} = C \cdot x_k + d$$

Jacobi Method: 1/4

- **Equation *i* in A•x=b has the following form:**

$$a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,i}x_i + \cdots + a_{i,n}x_n = b_i$$

- **Solving for x_i yields the following if $a_{i,i} \neq 0$:**

$$x_i = \frac{1}{a_{i,i}}\left[b_i - \left(a_{i,1}x_1 + \cdots + a_{i,i-1}x_{i-1} + a_{i,i+1}x_{i+1} + \cdots + a_{i,n}x_n\right)\right] = \frac{1}{a_{i,i}}\left[b_i - \sum_{k=1,k\neq i}^{n} a_{i,k}x_k\right]$$

- **Jacobi method goes as follows:**
  - **Start with an initial guess $\mathbf{x}_0 = [x_1, x_2, \ldots, x_n]$**
  - **Plug this $\mathbf{x}_i$ into the above equations to compute $\mathbf{x}_{i+1}$**
  - **Repeat this process until $\mathbf{A \bullet x}_k \approx \mathbf{b}$ for some *k*.**
- **See next few slides for more details.**

Jacobi Method: 2/4

- **Let the system be:**

$$\begin{aligned} -5x - y + 2z &= 1 \\ 2x + 6y - 3z &= 2 \\ 2x + y + 7z &= 32 \end{aligned} \quad \xrightarrow{\textit{transformation}} \quad \begin{aligned} x &= -(1 + y - 2z)/5 \\ y &= (2 - 2x + 3z)/6 \\ z &= (32 - 2x - y)/7 \end{aligned}$$

- **Suppose the initial values are $x=y=z=0$ (*i.e.*, $\mathbf{x}_0 = [0,0,0]$)**
- **Plug $\mathbf{x}_0$ into the equations: $\mathbf{x}_1=[-1/5,1/3,32/7]$.**
- **Plug $\mathbf{x}_1$ into the equations:**

$$x_2 = \begin{bmatrix} -\frac{1}{5}\left(1+\frac{1}{3}-2\times\frac{32}{7}\right) \\ \frac{1}{6}\left(2-2\times\frac{-1}{5}+3\times\frac{32}{7}\right) \\ \frac{1}{7}\left(32-2\times\frac{-1}{5}-\frac{1}{3}\right) \end{bmatrix} = \begin{bmatrix} -1.5619 \\ 2.6857 \\ 4.5810 \end{bmatrix}$$

Converge in approx. 10 iterations with $x\approx0.998$, $y\approx1.997$, $z\approx3.991$

Jacobi Method: 3/4

- Step 1: Initialization

```
! Given system is A•x = b
! C(:,:) and d(:) are two
!   working arrays

DO i = 1, n
  DO j = 1, n
    C(i,j) = -A(i,j)/A(i,i)
  END DO
  C(i,i) = 0
  d(i) = b(i)/A(i,i)
END DO
```

Now we have $x = C \bullet x + d$

$$C = \begin{bmatrix} 0 & -\frac{a_{1,2}}{a_{1,1}} & -\frac{a_{1,3}}{a_{1,1}} & \cdots & -\frac{a_{1,n}}{a_{1,1}} \\ -\frac{a_{2,1}}{a_{2,2}} & 0 & -\frac{a_{2,3}}{a_{2,2}} & \cdots & -\frac{a_{2,n}}{a_{2,2}} \\ -\frac{a_{3,1}}{a_{3,3}} & -\frac{a_{3,2}}{a_{3,3}} & 0 & \ddots & -\frac{a_{3,n}}{a_{3,3}} \\ \vdots & \vdots & \vdots & & \vdots \\ -\frac{a_{n,1}}{a_{n,n}} & -\frac{a_{n,2}}{a_{n,n}} & -\frac{a_{n,3}}{a_{n,n}} & \cdots & 0 \end{bmatrix}$$

$$d = \begin{bmatrix} \frac{b_1}{a_{1,1}} \\ \frac{b_2}{a_{2,2}} \\ \frac{b_3}{a_{3,3}} \\ \vdots \\ \frac{b_n}{a_{n,n}} \end{bmatrix}$$

row and column swaps may be needed before starting iteration!

Jacobi Method: 4/4

- Step 2: Iteration

```
DO
  x_new = C*x + d
  IF (ABS(A*x_new - b) < Tol) EXIT
  x = x_new
END DO
```

`ABS(A*x_new-b) < Tol` means the absolute value of each entry of **`A*x_new - b`** is less than **`Tol`** .

You may just use the equations for computation instead of the above matrix form.

Gauss-Seidel Method: 1/4

- **Gauss-Seidel method is slightly different from Jacobi method.**
- **With Jacobi method, all new x values are computed and be used in the *next* iteration.**
- **Gauss-Seidel method uses the same set of equations and the same initial values.**
- ***However*, the new x_1 computed from equation 1 is used in equation 2 to compute x_2, the new x_1 and x_2 are used in equation 3 to compute x_3.**
- **In general, the new x_1, x_2, …, x_{i-1} are used in equation i to compute the new x_i.**

Gauss-Seidel Method: 2/4

- **Example:**

$$\begin{aligned} -5x - y + 2z &= 1 \\ 2x + 6y - 3z &= 2 \\ 2x + y + 7z &= 32 \end{aligned} \xrightarrow{\textit{transformation}} \begin{aligned} x &= -(1 + y - 2z)/5 \\ y &= (2 - 2x + 3z)/6 \\ z &= (32 - 2x - y)/7 \end{aligned}$$

- **Initial value is $x=y=z=0$.**
- **Plugging y and z into equation 1 yields the new $x = -1/5$. Now, we have $x = -1/5$, $y=z=0$.**
- **Plugging x and z into equation 2 yields the new $y = 2/5$. Now, we have $x=-1/5$, $y=2/5$, $z=0$**
- **Plugging x and y into equation 3 yields the new $z = 32/7$. Now, we have $x=-1/5$, $y=2/5$, $z=32/7$.**
- **This completes the first iteration!**

Gauss-Seidel Method: 3/4

- **Example:**

$$\begin{aligned} -5x - y + 2z &= 1 \\ 2x + 6y - 3z &= 2 \\ 2x + y + 7z &= 32 \end{aligned} \xrightarrow{\text{transformation}} \begin{aligned} x &= -(1 + y - 2z)/5 \\ y &= (2 - 2x + 3z)/6 \\ z &= (32 - 2x - y)/7 \end{aligned}$$

- **Iteration 2 starts with $x=-1/5$, $y=2/5$, $z=32/7$.**
- **Plugging y and z into equation 1 yields the new $x = 271/175 \approx 1.5486$, and $x = 271/175$, $y=2/5$, $z=32/7$.**
- **Plugging x and z into equation 2 yields the new $y = 368/175 \approx 2.1029$, and $x=271/175$, $y=368/175$, $z=32/7$**
- **Plugging x and y into equation 3 yields the new $z = 134/35 \approx 3.8286$, and $x=271/175$, $y=368/175$, $z=134/35$.**
- **This completes the second iteration!**

Gauss-Seidel Method: 4/4

- Algorithm:
- **The initialization part is the same as the Jacobi method, since both methods use the same set of equation transformations.**

```
DO
   DO i = 1, n                    ! Update x_i
      EQN_i = 0
      DO j = 1, n
         EQN_i = EQN_i + C(i,j)*x(j)
      END DO
      x(i) = EQN_i + d(i)
   END DO
   IF (ABS(A*x - b) < Tol) EXIT
END DO
```

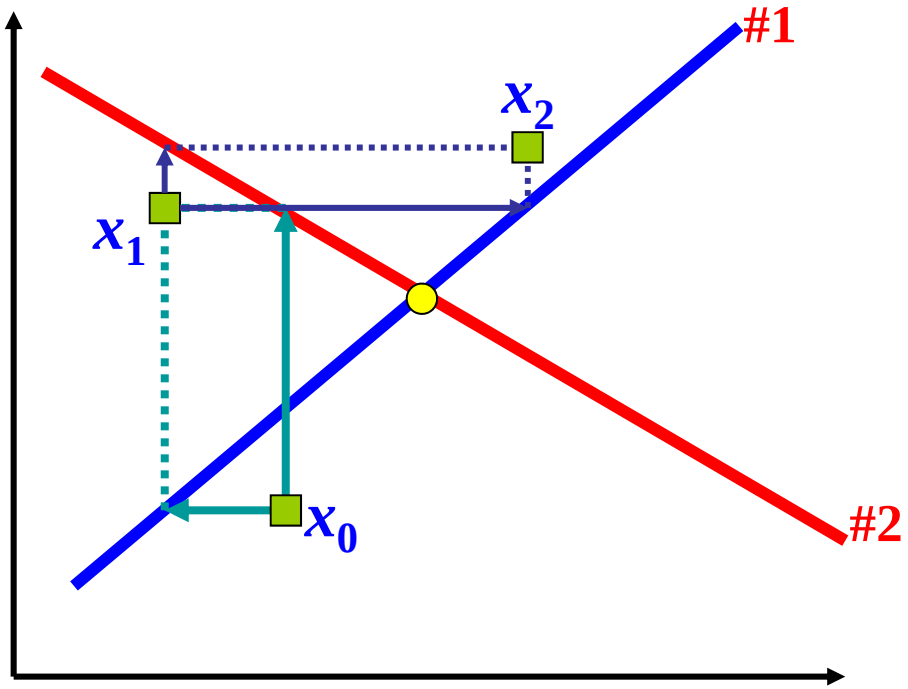
Convergence

- **Jacobi and Gauss-Seidel methods may not converge.**
- **Even though they may converge, they may be very slow.**
- **If the system is *diagonal dominant*, both methods converge.**

$$|a_{i,i}| > \sum_{j=1, j\neq i}^{n} |a_{i,j}| \quad \textbf{(for every } i\textbf{)}$$

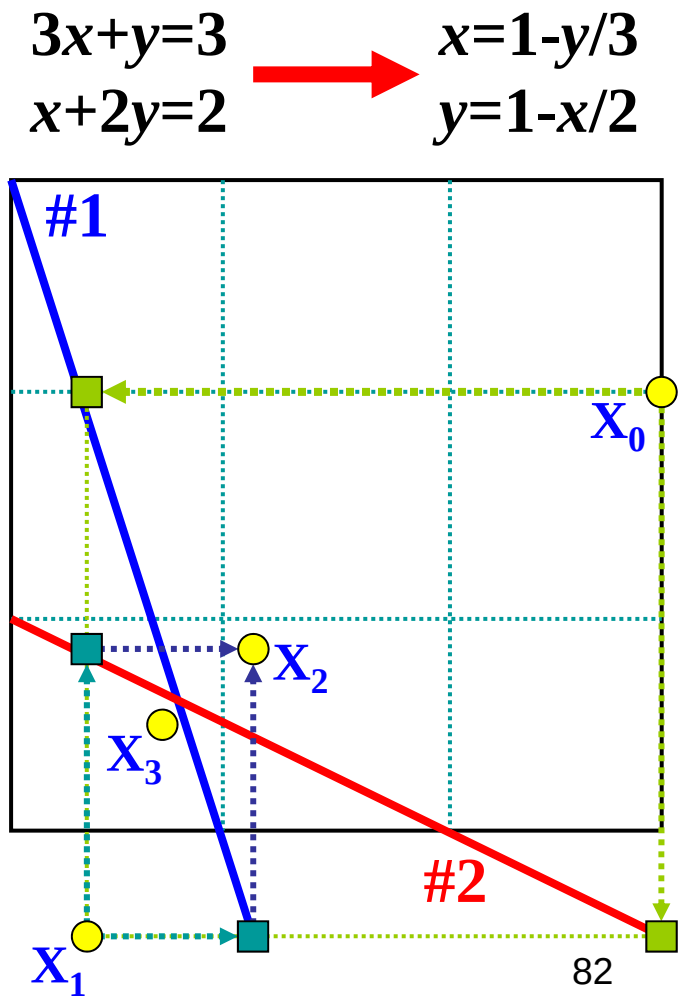
Geometric Meaning: 1/4

- **Suppose we have two lines, #1 and #2.**
- **Jacobi method uses *y* to find a new *x* with equation #1, and uses *x* to find a new *y* with equation #2.**



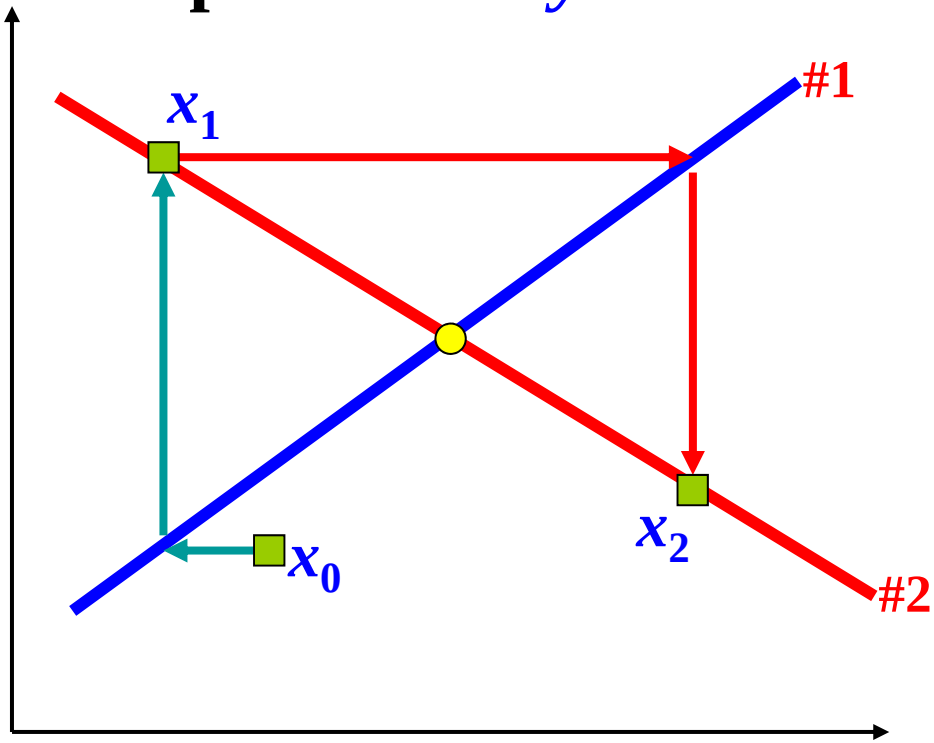
Geometric Meaning: 2/4

- If the given system is *diagonal dominant*, the Jacobi method converges.
- Let $X_0=[3,2]$.
- Then, $X_1=[1/3,-1/2]$. The right diagram shows how to find the *x* and *y* coordinates.
- $X_2=[7/6,5/6]$ and $X_3=[13/18,5/12]$.
- The solution is $X^*=[4/5,3/5]$



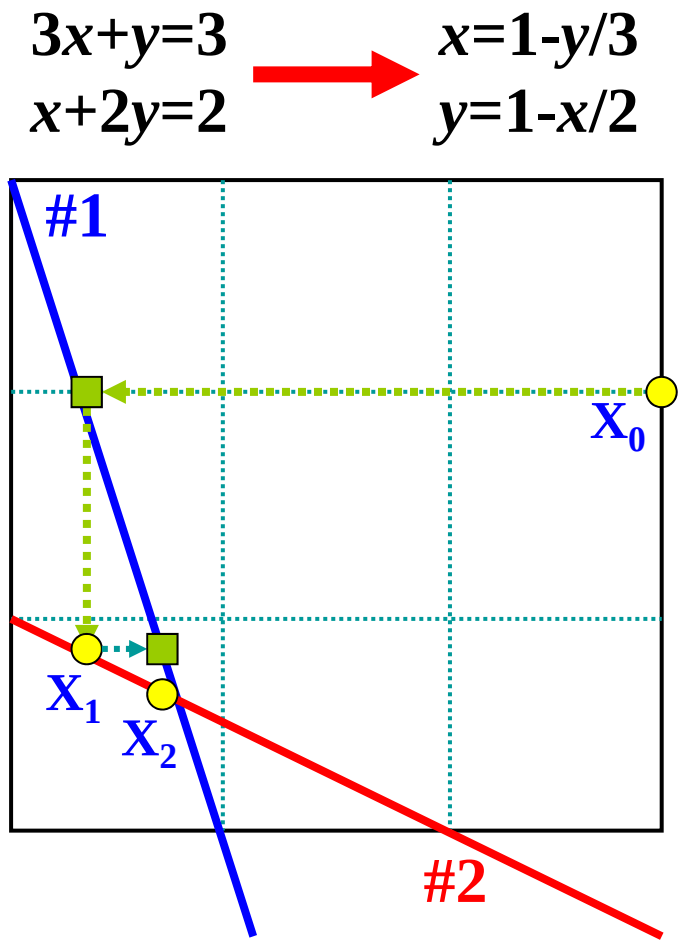
Geometric Meaning: 3/4

- **Let us try the Gauss-Seidel method.**
- **This method uses *y* to compute a new *x* with equation 1, which replaces the old *x*, and uses this new *x* to compute a new *y*.**



Geometric Meaning: 4/4

- **Use Gauss-Seidel method with $X_0=[3,2]$.**
- **Since $x=1-2/3=1/3$ and $y=1-(1/3)/2=5/6$, $X_1=[1/3,5/6]$.**
- **Since $x=1-(5/6)/3=13/18$ and $y=1-(13/18)/2=23/36$, $X_2=[13/18,23/36]$.**
- **Since $x=1-(23/36)/3=85/108$ and $y=1-(85/108)/2=131/216$, $X_3=[85/108,131/216]$.**
- **Gauss-Seidel is *faster*!**



Iterative Refinement: 1/5

- **Due to rounding error accumulation, elimination methods may not be accurate enough.**
- **This is especially true if matrix A in A·x = b is ill-conditioned (*e.g.*, nearly singular), and, as a result, the computed solution may still be far away from the actual solution.**
- ***Iterative refinement techniques* can improve the accuracy of elimination methods.**
- **To use iterative refinement methods, one has to preserve matrix A and computes the LU-decomposition A = L·U.**

Iterative Refinement: 2/5

- **Here is the algorithm. In each step, the "residual" is computed and used to solve for a "correction" to update the solution X.**

```
Make copies of A to A' and A" and b to b'
Use A" to compute A" = L*U      ! A" is destroyed
Apply an elimination method to solve A'*x=b'
Let the solution be x
DO
  r = b - A * x                 ! compute residual
  IF (ABS(r) < Tol) EXIT
  Solve for Δ from (L*U)*Δ = r  ! compute correction
  x = x + Δ                     ! update x
END DO
```

Iterative Refinement: 3/5

- **Consider solving the following system:**

$$x + y = 2$$

$$2x + 3y = 5$$

- **We have A, L, U and b matrices as follows:**

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$$

L **U**

- **If a method provides a "solution" $x = 0.9$ and $y = 1.3$, the *residual* vector r is**

$$r = b - A \cdot x = \begin{bmatrix} 2 \\ 5 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 0.9 \\ 1.3 \end{bmatrix} = \begin{bmatrix} -0.2 \\ -0.7 \end{bmatrix}$$

Iterative Refinement: 4/5

- **Now, we solve for Δ from A·Δ=r.**
- **Since A=L·U, we have L·(U·Δ)=r.**
- **Since L and r are known, we may solve for T in L·T = r (hence U·Δ = T) as follows:**

$$\underbrace{\begin{bmatrix} 1 & \\ 2 & 1 \end{bmatrix}}_{\mathbf{L}} \cdot \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} -0.2 \\ -0.7 \end{bmatrix}$$

- **Forward substitution yields t_1 = -0.2 and t_2 = -0.3**

$$T = \begin{bmatrix} -0.2 \\ -0.3 \end{bmatrix}$$

Iterative Refinement: 5/5

- **We have $U \cdot \Delta = T$ as follows:**

$$\underbrace{\begin{bmatrix} 1 & 1 \\ & 1 \end{bmatrix}}_{U} \cdot \begin{bmatrix} \Delta_1 \\ \Delta_2 \end{bmatrix} = \begin{bmatrix} -0.2 \\ -0.3 \end{bmatrix}$$

- **Backward substitution yields Δ as follows:**

$$\Delta = \begin{bmatrix} 0.1 \\ -0.3 \end{bmatrix}$$

- **The new x is**

$$x_{new} = x_{old} + \Delta = \begin{bmatrix} 0.9 \\ 1.3 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

- **Since $A \cdot x = b$, we have the solution in 1 iteration!**

Matrix Inversion: 1/9

- **The inverse matrix of a $n \times n$ matrix $\mathbf{A}$, $\mathbf{A}^{-1}$, satisfies $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{A}^{-1} \cdot \mathbf{A} = \mathbf{I}$, where $\mathbf{I}$ is the $n \times n$ identity matrix.**
- **Not all matrices are invertible. A matrix that does not an inverse is a singular matrix, and has zero determinant.**
- Given $\mathbf{A}$, how do we find its inverse?
- **If you know how to solve systems of linear equations, matrix inversion is just a little more complex.**

Matrix Inversion: 2/9

- Let $\mathbf{X}$ be a matrix such that $\mathbf{A} \cdot \mathbf{X} = \mathbf{I}$.
- Consider column i of $\mathbf{X}$, $\mathbf{X}_i$, and column i of $\mathbf{I}$, $\mathbf{I}_i$.
- We have $\mathbf{A} \cdot \mathbf{X}_i = \mathbf{I}_i$ as shown below.
- Since $\mathbf{X}_i$ is unknown and $\mathbf{I}_i$ is known, solving for $\mathbf{X}_i$ gives column i of matrix $\mathbf{X}$.

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,i} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots & & \vdots \\ a_{i,1} & & a_{i,i} & & a_{i,n} \\ \vdots & & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \cdot \begin{bmatrix} x_{1,i} \\ \vdots \\ x_{i,i} \\ \vdots \\ x_{n,i} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

***i*-th position**

Matrix Inversion: 3/9

- **Therefore, we may use the same A and solve X_1 from I_1, X_2 from I_2, …, X_n from I_n.**
- **It requires the execution of a linear system solver *n* times with the same A.**
- **We may use LU-decomposition to save time!**

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,i} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots & & \vdots \\ a_{i,1} & & a_{i,i} & & a_{i,n} \\ \vdots & & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \cdot \begin{bmatrix} x_{1,i} \\ \vdots \\ x_{i,i} \\ \vdots \\ x_{n,i} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

***i*-th position**

Matrix Inversion: 4/9

- First, do a LU-decomposition $A = L \cdot U$.
- For each i from 1 to n, let I_i be the column vector with a 1 in the i-th position and 0 elsewhere.
- We have $(L \cdot U) \cdot X_i = I_i$. Applying forward and backward substitutions yields X_i, the i-th column of the inverse matrix $X = A^{-1}$.
- Note that if partial (complete) pivoting is used in the construction of $A = L \cdot U$, one must swap rows of I_k's before solving, and swap rows (and columns) after the solution is obtained.

Matrix Inversion: 5/9

- **Consider finding the inversion of the following matrix A:**

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

- **First, find A's LU-decomposition as follows:**

$$A = L \cdot U = \begin{bmatrix} 1 & & \\ \frac{3}{4} & 1 & \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 4 & 0 & 1 \\ & 1 & \frac{9}{4} \\ & & \frac{-1}{4} \end{bmatrix}$$

Matrix Inversion: 6/9

- **Now, find the first column of the inverse.**
- **The right-hand side is $[1,0,0]^T$ as follows:**

$$\begin{bmatrix} 1 & & \\ \frac{3}{4} & 1 & \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 4 & 0 & 1 \\ & 1 & \frac{9}{4} \\ & & \frac{-1}{4} \end{bmatrix} \cdot X_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

- **Go through a forward and a backward substitution yields:**

$$X_1 = \begin{bmatrix} 1 \\ 6 \\ -3 \end{bmatrix}$$

Matrix Inversion: 7/9

- **Then, find the second column of the inverse.**
- **The right-hand side is [0,1,0]^T as follows:**

$$\begin{bmatrix} 1 & & \\ \frac{3}{4} & 1 & \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 4 & 0 & 1 \\ & 1 & \frac{9}{4} \\ & & \frac{-1}{4} \end{bmatrix} \cdot X_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- **Go through a forward and a backward substitution yields:**

$$X_2 = \begin{bmatrix} -1 \\ -8 \\ 4 \end{bmatrix}$$

Matrix Inversion: 8/9

- **Finally, find the third column of the inverse.**
- **The right-hand side is $[0,0,1]^T$ as follows:**

$$\begin{bmatrix} 1 & & \\ \frac{3}{4} & 1 & \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 4 & 0 & 1 \\ & 1 & \frac{9}{4} \\ & & \frac{-1}{4} \end{bmatrix} \cdot X_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

- **Go through a forward and a backward substitution yields:**

$$X_3 = \begin{bmatrix} 1 \\ 9 \\ -4 \end{bmatrix}$$

Matrix Inversion: 9/9

- **Therefore, the inverse of A is $[X_1 \mid X_2 \mid X_3]$:**

$$A^{-1} = \begin{bmatrix} 1 & -1 & 1 \\ 6 & -8 & 9 \\ -3 & 4 & -4 \end{bmatrix}$$

- **Let us verity it:**

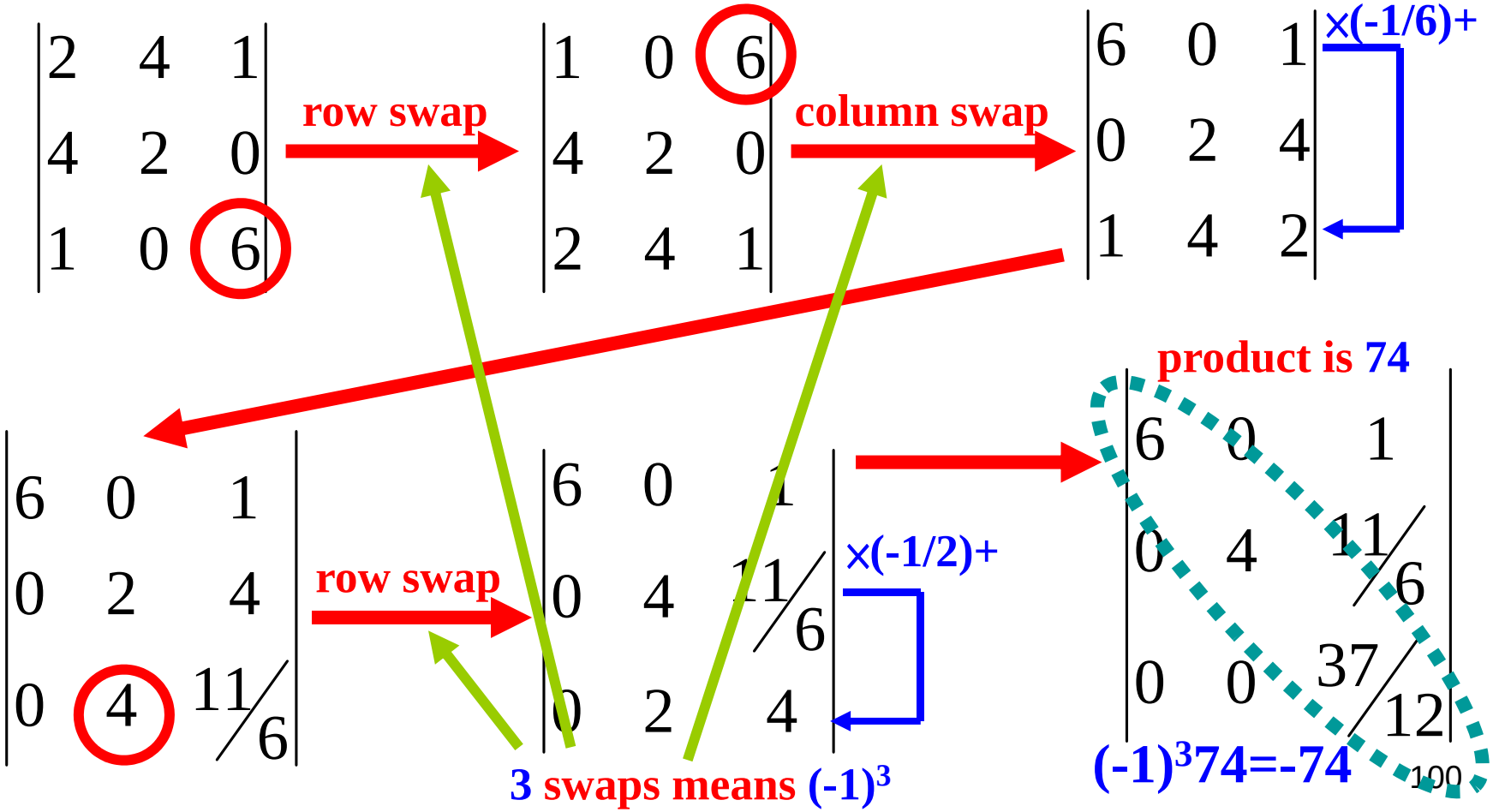
$$A \cdot A^{-1} = \begin{bmatrix} 4 & 0 & 1 \\ 3 & 1 & 3 \\ 0 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 & 1 \\ 6 & -8 & 9 \\ -3 & 4 & -4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Determinant: 1/3

- **The determinant of a square matrix is also easy to compute.**
- **If $A = L \cdot U$, then the determinant of A is the product of the diagonal elements of U.**
- **If the construction of $A = L \cdot U$ uses pivoting, the total number of row and column swaps matters. If the total is odd, the product should be multiplied by -1.**
- **This is because swapping two rows (or columns) changes the sign of the determinant.**

Determinant: 2/3

- **Here is an example with complete pivoting.**



Determinant: 3/3

- **It is possible that all the remaining entries are 0's when doing complete pivoting.**
- **In this case, the given matrix is *singular* with zero determinant.**
- **More importantly, the number of none-zero entries on the diagonal is the *rank* of the matrix.**

The End