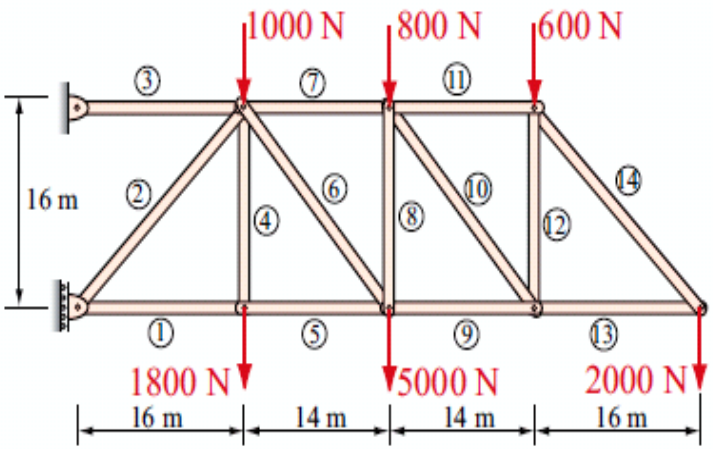
Linear System of Equations

- Linear systems are perhaps the most widely applied numerical procedures when real-world situation are to be simulated.
- Example: computing the forces in a TRUSS.



$$
\begin{aligned}
-F_1 + F_5 &= 0 \\
-0.71071F_2 - F_3 + 0.6585F_6 + F_7 &= 0 \\
-0.7071F_2 - F_4 - 0.7525F_6 &= 1000 \\
F_4 &= 1800 \\
-F_7 + 0.6585F_{10} + F_{11} &= 0 \\
-F_8 - 0.7525F_{10} &= 800 \\
-F_5 - 0.6585F_6 + F_9 &= 0 \\
0.7525F_6 + F_8 &= 5000 \\
-F_{11} + 0.7071F_{14} &= 0 \\
-F_{12} - 0.7071F_{14} &= 600 \\
-F_9 - 0.6585F_{10} + F_{13} &= 0 \\
0.7525F_{10} + F_{12} &= 0 \\
-F_{13} + 0.7071F_{14} &= 0 \\
0.7071F_{14} &= 2000
\end{aligned}
$$

- 14 equations with 14 unknowns

- Methods for numerically solving ordinary and partial differential equations depend on Linear Systems of Equations.
- No of unknowns = number of equations i.e. n×n system

Linear System of Equations

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

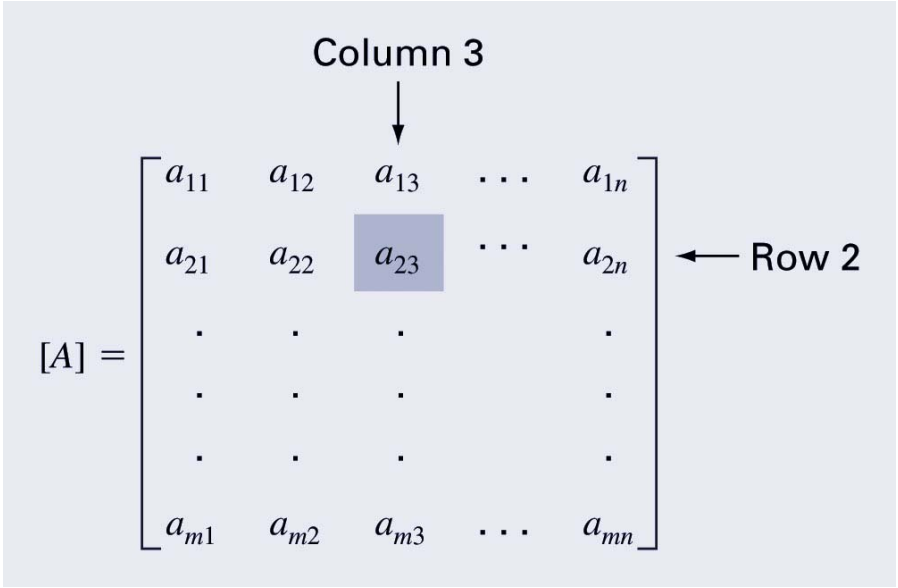
$$A\,x = b$$

- n * n Matrix
- Column vector
- Column vector

Matrix Notation

- A *matrix* consists of a rectangular array of elements represented by a single symbol (example: [*A*]).

- An individual entry of a matrix is an *element* (example: a_{23})



Matrix Notation

- A horizontal set of elements is called a *row* and a vertical set of elements is called a *column*.
- The first subscript of an element indicates the row while the second indicates the column.
- The size of a matrix is given as *m* rows by *n* columns, or simply *m* by *n* (or *m* x *n*).
- 1 x *n* matrices are *row vectors*.
- *m* x 1 matrices are *column vectors*.

Special Matrices

- Matrices where $m = n$ are called *square matrices*.

- There are a number of special forms of square matrices:

Symmetric

$$[A] = \begin{bmatrix} 5 & 1 & 2 \\ 1 & 3 & 7 \\ 2 & 7 & 8 \end{bmatrix}$$

Diagonal

$$[A] = \begin{bmatrix} a_{11} & & \\ & a_{22} & \\ & & a_{33} \end{bmatrix}$$

Identity

$$[A] = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}$$

Upper Triangular

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ & a_{22} & a_{23} \\ & & a_{33} \end{bmatrix}$$

Lower Triangular

$$[A] = \begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Banded

$$[A] = \begin{bmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & a_{43} & a_{44} \end{bmatrix}$$

Matrix Operations

- Two matrices are considered **equal** if and only if every element in the first matrix is equal to every corresponding element in the second. This means the two matrices must be the same size.

$$A=B \text{ if } a_{ij}= b_{ij} \text{ for all i and j}$$

- Matrix **addition and subtraction** are performed by adding or subtracting the corresponding elements. This requires that the two matrices be the same size.

$$C=A+B \quad \Rightarrow \quad c_{ij}=a_{ij}+ b_{ij}$$

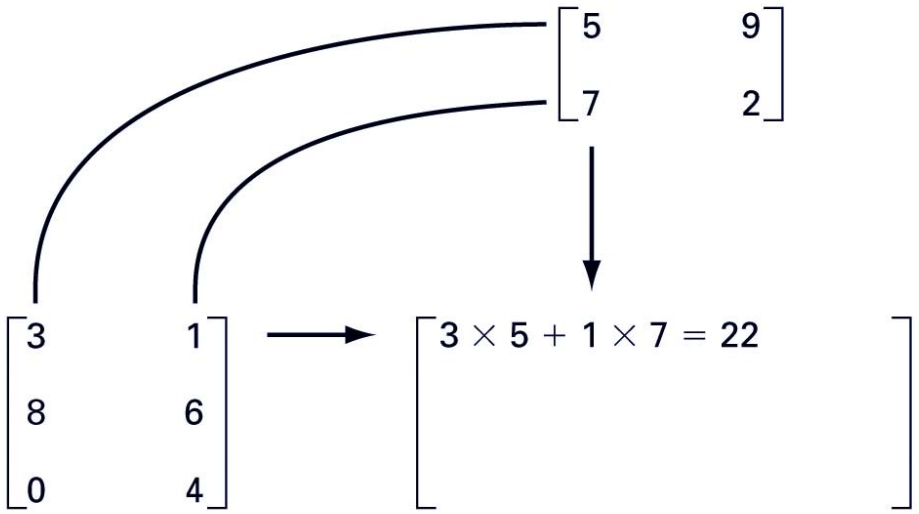
- **multiplication** by a scalar is performed by multiplying each element by the same scalar.

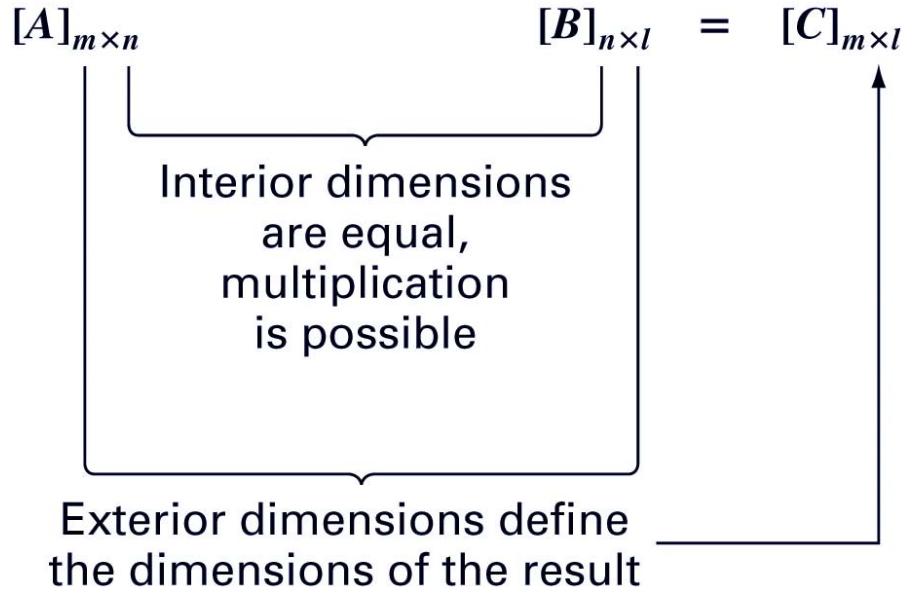
$$D=gA \quad \Rightarrow \quad \begin{vmatrix} ga_{11} & ga_{12} & ga_{13} \\ ga_{21} & ga_{22} & ga_{23} \\ ga_{31} & ga_{32} & ga_{33} \end{vmatrix}$$

Matrix Multiplication

- The elements in the matrix [C] that results from multiplying matrices [A] and [B] are calculated using:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$





Matrix Inverse and Transpose

- The *inverse* of a square, nonsingular matrix $[A]$ is that matrix which, when multiplied by $[A]$, yields the identity matrix.

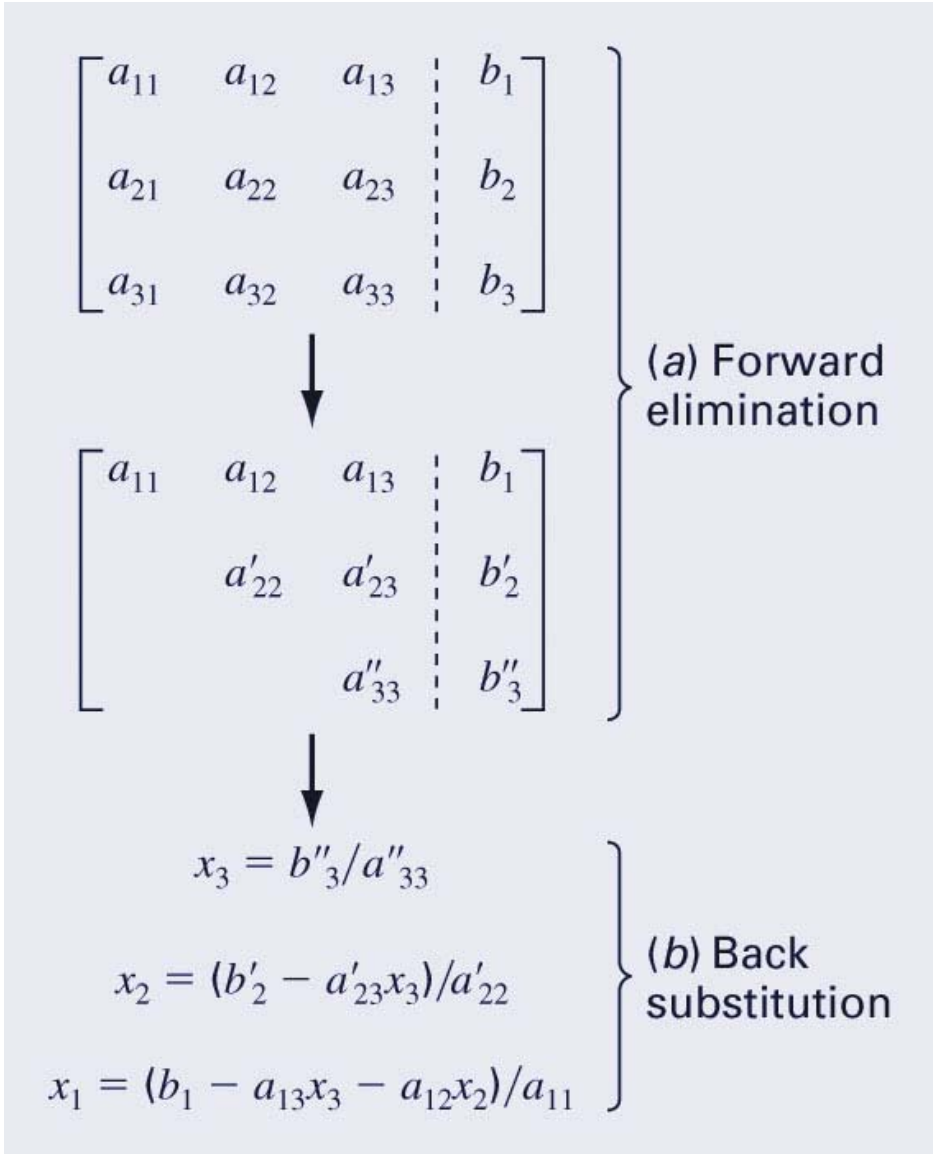
$$[A][A]^{-1}=[A]^{-1}[A]=[I]$$

- The *transpose* of a matrix involves transforming its rows into columns and its columns into rows.

$$(a_{ij})^{T}=a_{ji}$$

Gauss Elimination method

- Forward elimination
  - Starting with the first row, add or subtract multiples of that row to eliminate the first coefficient from the second row and beyond.
  - Continue this process with the second row to remove the second coefficient from the third row and beyond.
  - Stop when an upper triangular matrix remains.
- Back substitution
  - Starting with the *last* row, solve for the unknown, then substitute that value into the next highest row.
  - Because of the upper-triangular nature of the matrix, each row will contain only one more unknown.



Gauss Elimination method

```
clc; close all; format long e;
% A= coefficient matrix % b=right hand side vector
% x=solution vector
A=[10 -7 0; -3 2 6; 5 -1 5]; b=[7;4;6];
[m,n]=size(A);
if m~=n
    error('Matrix A must be square');
end
Nb=n+1;
Aug=[A b]; % Augmented matrix, combines A and b
%----forward elimination-------
for k=1:n-1 % from row 1 to n-1
    if abs(A(k,k))<eps
        error('Zero Pivot encountered')
    end
    for j=k+1:n
        multiplier=Aug(j,k)/Aug(k,k);
        Aug(j,k:Nb)=Aug(j,k:Nb)-multiplier*Aug(k,k:Nb);
    end
end
```

Gauss Elimination method

```
%----Back substitution---------
x=zeros(n,1);
x(n)=Aug(n,Nb)/Aug(n,n);

for k=n-1:-1:1
    for j=k+1:n
        Aug(k,Nb)=Aug(k,Nb)-Aug(k,j)*x(j);
                                  % computes b(k)-sum(akj*xj)
    end

     x(k)=Aug(k,Nb)/Aug(k,k);
end
```

Gauss Elimination method

• The execution of Gauss elimination depends on the amount of *floating-point operations* (or flops). The flop count for an *n* x *n* system is:

Forward Elimination	$\frac{2n^3}{3} + O(n^2)$
Back Substitution	$n^2 + O(n)$
Total	$\frac{2n^3}{3} + O(n^2)$

Gauss Elimination method

Number of flops for Gauss Elimination method:

n	Total flops	Forward Elimination	Back substitution	$2n^3/3$	% due to Elimination
10	805	705	100	667	87.58%
100	681550	671550	10^4	666667	98.53%
1000	$6.68*10^8$	$6.67*10^8$	10^6	$6.67*10^8$	99.85%

- As the system gets larger, the cost of computing increases
  - Flops increases nearly 3 orders of magnitude for every order of increase in the number of equations
- Most of the efforts is incurred in the forward elimination steps.

Cost of computing

Example 1:

Estimate the time required to carry out back substitution on a system of 500 equations in 500 unknowns, on a computer where elimination takes 1 second.

For Elimination

approx. flops, $fe = 2n^3/3 = 2*500^3/3$

time needed, te=1 sec

For Back substitution

approx. flops, $fb = n^2 = 500^2$

time needed, tb=?

$tb/te = fb/fe$

$tb = te*fb/fe = 3/(2*500) = 0.003$ sec

Back substitution time=0.003 sec

Total computation time=1.003 sec

Cost of computing

Example 2:

Assume that your computer can solve a 200*200 linear system Ax=b in 1 second by Gauss elimination method. Estimate the time required to solve four systems of 500 equations in 500 unknowns with the same coefficient matrix, using LU factorization method.

For Gauss Elimination

approx. flops, $fg = 2n^3/3 = 2*200^3/3$

time needed, tg=1 sec

For LU

approx. flops, $fl = 2n^3/3 + 2kn^2 = 2*500^3/3 + 2*4*500^2$

time needed,

$tl = tg*fl/fg = (2*500\text{^}3/3 + 2*4*500^2)/(2*200^3/3) = 16$ sec

If we do the same 4 problems by Gauss Elimination:

$tg = 4*500^3/200^3 = 62.5$ sec

Time save=46.5 sec ≈ 300% !!!

Forward and backward error

Definition

Let x_c be an approximate solution of the linear system $Ax=b$.

The residual is the vector $r=b-Ax_c$

The backward error = $||b-Ax_c||_\infty$

The forward error = $||x_{exact}-x_c||_\infty$

Error magnification factor = $\dfrac{\text{relative forward error}}{\text{relative backward error}} = \dfrac{\dfrac{\|x_{exact}-x_c\|_\infty}{\|x_{exact}\|_\infty}}{\dfrac{\|b-Ax_c\|_\infty}{\|b\|_\infty}}$

Sources of error

There are two major sources of error in Gauss-elimination as we have discussed so far.

- ill-conditioning
- swamping

Ill-conditioning

-sensitivity of the solution to the input data.

1.First we examine the effect of small change in the coefficients

Let us consider a 2×2 system: Ax=b

$$\begin{bmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 2.00 \\ 2.00 \end{Bmatrix}$$

Exact solution $x_{exact} = \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$

Now, we make a small change in the coefficient matrix A

$$A_1 = \begin{bmatrix} 1.01 & 0.98 \\ 0.99 & 1.01 \end{bmatrix}$$

$\Rightarrow$ New solution $x_{C1} = \begin{Bmatrix} 1.2 \\ 0.8 \end{Bmatrix}$

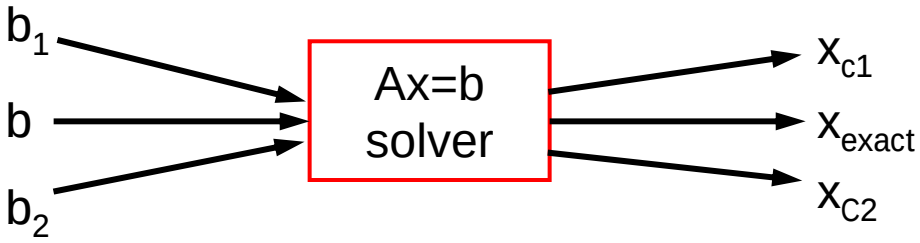
Large change in soln

Sources of error: Ill-conditioning

2. we examine the effect of small change in 'b' for the same coefficient matrix A

$$b_1 = \begin{Bmatrix} 2.02 \\ 1.98 \end{Bmatrix} \quad \Rightarrow \text{solution} \quad x_{C1} = \begin{Bmatrix} 2 \\ 0 \end{Bmatrix}$$

$$b_2 = \begin{Bmatrix} 1.98 \\ 2.02 \end{Bmatrix} \quad \Rightarrow \text{solution} \quad x_{C2} = \begin{Bmatrix} 0 \\ 2 \end{Bmatrix}$$



Here we see, even though the 3 inputs are "close together"
we get very distinct outputs

Therefore, for ill-conditioned system,

small change in input, we get large change in output.

How can we express this characteristic mathematically?

Sources of error: Ill-conditioning

Let us consider another example of ill-conditioned system:

$$\begin{bmatrix} 1 & 1 \\ 1.0001 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 2 \\ 2.0001 \end{Bmatrix} \qquad x_{exact} = \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} \qquad x_C = \begin{Bmatrix} -1 \\ 3.0001 \end{Bmatrix}$$

Backward error vector = $r = b - Ax_C = \begin{Bmatrix} -0.0001 \\ +0.0001 \end{Bmatrix}$ $\|r\|_\infty = \|b - Axc\|_\infty = 0.0001$

Forward error vector = $x_{exact} - x_C = \begin{Bmatrix} 2 \\ -2.0001 \end{Bmatrix}$ $\|x_{exact} - x_c\|_\infty = 2.0001$

Relative backward error = $\dfrac{\| b - Ax_C \|_\infty}{\| b \|_\infty} = \dfrac{0.0001}{2.0001} = 0.005\%$

Relative forward error = $\dfrac{\| x_{exact} - x_C \|_\infty}{\| x_{exact} \|_\infty} = 2.0001 \approx$ **200%**

For ill-conditioned system, error magnification factor is very high

Error magnification factor = 2.0001/(0.0001/2.0001) = **40004.001**

Condition number

There is another quantity known as “Condition number” is also used to show that the system is ill-conditioned.

Definition

The condition number of a square matrix A, cond(A), is the maximum possible error magnification factor for Ax=b, over all right-hand side b.

Theorem

The condition number of a n*n matrix A is $cond(A)=||A||*||A^{-1}||$

- **For ill-conditioned system, condition number is high**

If the coefficient matrix A have t-digit precision, and $cond(A)=10^k$ The solution vector x is accurate upto (t-k) digits.

Example: elements of a coefficient matrix A is accurate up to 5 digits, and $cond(A)=10^4$

So, the solution is accurate (5-4) = 1 digit only !!!

Sources of error: Swamping

- A second significant source of error in Gauss elimination method is Swamping.
- Much easier to fix.

- The Gauss elimination method discussed so far, works well as long as the diagonal elements of coefficient matrix A (pivot) are non-zero.
- Note:
  - –The computation of the multiplier and the back substitution require divisions by the pivots.
  - –Consequently the algorithm can not be carried out if any of the pivots are zero or near to zero.
- Let us consider our original example again,

$$\begin{bmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7 \\ 4 \\ 6 \end{Bmatrix} \Rightarrow \begin{bmatrix} 10 & -7 & 0 \\ -3 & 2.099 & 6 \\ 5 & -1 & 5 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7 \\ 3.901 \\ 6 \end{Bmatrix} \qquad x_{exact} = \begin{Bmatrix} 0 \\ -1 \\ 1 \end{Bmatrix}$$

Sources of error: Swamping

- let us assume that the solution is to be computed on a hypothetical machine that does decimal floating point arithmetic with five significant digits.
- the first step of elimination produces,

$$R_2' = R_2 - R_1\left(\frac{-3}{10}\right)$$

$$R_3' = R_3 - R_1\left(\frac{5}{10}\right)$$

$$\Rightarrow \quad \begin{bmatrix} 10 & -7 & 0 \\ 0 & -0.001 & 6 \\ 0 & 2.5 & 5 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7 \\ 6.001 \\ 2.5 \end{Bmatrix}$$

quite small compared with the other elements

Sources of error: Swamping

- without interchanging the rows, the multiplier $= \dfrac{2.5}{0.001} = 2.5 \times 10^3$

$$R_3'' = R_3' - R_2'\left(2.5 \times 10^3\right) \quad \Rightarrow$$

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & -0.001 & 6 \\ 0 & 0 & 5 + 2.5 \times 10^3 \times 6 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7 \\ 6.001 \\ 2.5 + 2.5 \times 10^3 \times 6.001 \end{Bmatrix}$$

Finally, $x_{C2} = \begin{Bmatrix} -0.35 \\ -1.5 \\ 0.99993 \end{Bmatrix}$ But, $x_{exact} = \begin{Bmatrix} 0 \\ -1 \\ 1 \end{Bmatrix}$

Sources of error: Swamping

- So, where did things go wrong?

- There is no "accumulation of rounding error" caused by doing thousands of arithmetic operations
- The matrix is NOT close to singular
- Actually,
  - The difficulty comes from choosing a small pivot at second step of the elimination.
  - As a result the multiplier is 2.5×10^3 and the final equation involves coefficients that are 10^3 times larger those in the original problem.
  - That means the equation is overpowered/swamped.
- If the multipliers are all less than or equal to 1 in magnitude, then computed solution can be proved to be satisfactory.
- Keeping the multipliers less than one in absolute value can be ensured by a process known as Pivoting.

Sources of error: Pivoting

- If we allow pivoting (partial), i.e, by interchanging R_2 and R_3, we have,

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.001 & 6 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7 \\ 2.5 \\ 6.001 \end{Bmatrix}$$

$$R_3' = R_3 - R_1\left(\frac{-0.001}{2.5}\right) \qquad x_{C1} = \begin{Bmatrix} -0.00042 \\ -1.0006 \\ 1.0003 \end{Bmatrix}$$

- Solution is close to the exact!!!

Ref: http://www.mathworks.com/moler/chapters.html

Pivoting

- Problems arise with Gauss elimination if a coefficient along the diagonal (Pivot element) is 0 (problem: division by 0) or close to 0 (problem: round-off error)

- One way to combat these issues is to determine the coefficient with the largest absolute value in the column below the pivot element. The rows can then be switched so that the largest element is the pivot element. This is called *partial pivoting.*

- If the rows to the right of the pivot element are also checked and columns switched, this is called *complete pivoting*.

Gauss Elimination with partial pivoting

```
clc; close all; format long e;
% A= coefficient matrix % b=right hand side vector
% x=solution vector
A=[10 -7 0; -3 2 6; 5 -1 5]; b=[7;4;6];
[m,n]=size(A);
if m~=n, error('Matrix A must be square');end
Nb=n+1;
Aug=[A b]; % Augmented matrix, combines A and b
%----forward elimilation-------
for k=1:n-1 % from row 1 to n-1
    %-----------partial pivoting------------
    [big, big_pos]=max(abs(Aug(k:n,k)));
    pivot_pos=big_pos+k-1;
    if pivot_pos~=k
       Aug([k pivot_pos],:)=Aug([pivot_pos k],:);
    end
    %---------------------------------------
    for j=k+1:n
        multiplier=Aug(j,k)/Aug(k,k);
        Aug(j,k:Nb)=Aug(j,k:Nb)-multiplier*Aug(k,k:Nb);
    end
end
```

Gauss Elimination method with partial pivoting

```
%----Back substitution---------
x=zeros(n,1);
x(n)=Aug(n,Nb)/Aug(n,n);

for k=n-1:-1:1
    for j=k+1:n
        Aug(k,Nb)=Aug(k,Nb)-Aug(k,j)*x(j);
                                % computes b(k)-sum(akj*xj)
    end

     x(k)=Aug(k,Nb)/Aug(k,k);
end
```

Scaling

Scaling is the operation of adjusting the coefficients of a set of equations so that they are all of the same order of magnitude.

Example $\begin{bmatrix} 3 & 2 & 100 \\ -1 & 3 & 100 \\ 1 & 2 & -1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 105 \\ 102 \\ 2 \end{Bmatrix}$ $\quad x_{exact} = \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix}$

If we do pivoting without scaling than round-off error may occur. For the above system, computed solution is $x_{C1} = \begin{Bmatrix} 0.94 \\ 1.09 \\ 1.00 \end{Bmatrix}$

But if you scale the system (divide the maximum coefficient in each Row) & then do the pivoting,

$R_1' = \frac{R_1}{100}$
$R_2' = \frac{R_2}{100}$
$R_3' = \frac{R_3}{2}$

$\Rightarrow \begin{bmatrix} 0.03 & 0.02 & 1 \\ -0.01 & 0.03 & 1 \\ 0.5 & 1 & -0.5 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1.05 \\ 1.02 \\ 1 \end{Bmatrix}$ $\quad \Rightarrow x_{C2} = \begin{Bmatrix} 1.00 \\ 1.00 \\ 1.00 \end{Bmatrix}$

Scaling

- So, whenever the coefficients in one column are widely different from those in another column, scaling is beneficial.
- When all values are about the same order of magnitude, scaling should be avoided, as additional round-off error incurred during the scaling operation itself may adversely affect the accuracy.

Matrix inversion

- For square matrix
- If matrix A is square, there is another matrix A-1, called the inverse of A, for which $AA^{-1}=I=A^{-1}A$

- Inverse can be computed in a column-by-column fashion by generating solutions with the unit vectors as the right-hand-side constants.

- Example: for a 3×3 system $b_1 = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}$

- Solution of the system $Ax_1=b_1$ will provide the column 1 of A^{-1}
- Then use, $b_2 = \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix}$ $b_3 = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$
- Solution of the system $Ax_2=b_2$ will provide the column 2 of A^{-1}
- Solution of the system $Ax_3=b_3$ will provide the column 3 of A^{-1}

Matrix inversion

- finally,

$$A^{-1} = \begin{array}{|ccc|} x_1 & x_2 & x_3 \\ ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{array}$$

- our original system is Ax = b $\Rightarrow$ x=A^{-1} * b

No of flops:

multiplication: $\frac{5}{6}n^3 + 2n^2 - \frac{5}{6}n$ addition: $\frac{4}{3}n^3 - \frac{1}{2}n^2 - \frac{5}{6}n$

Total flops $\approx \frac{16}{3}n^3 \approx 2n^3$

-no of flops 3 times Gauss elimination, $\Rightarrow$ more round-off error

-computationally more expensive

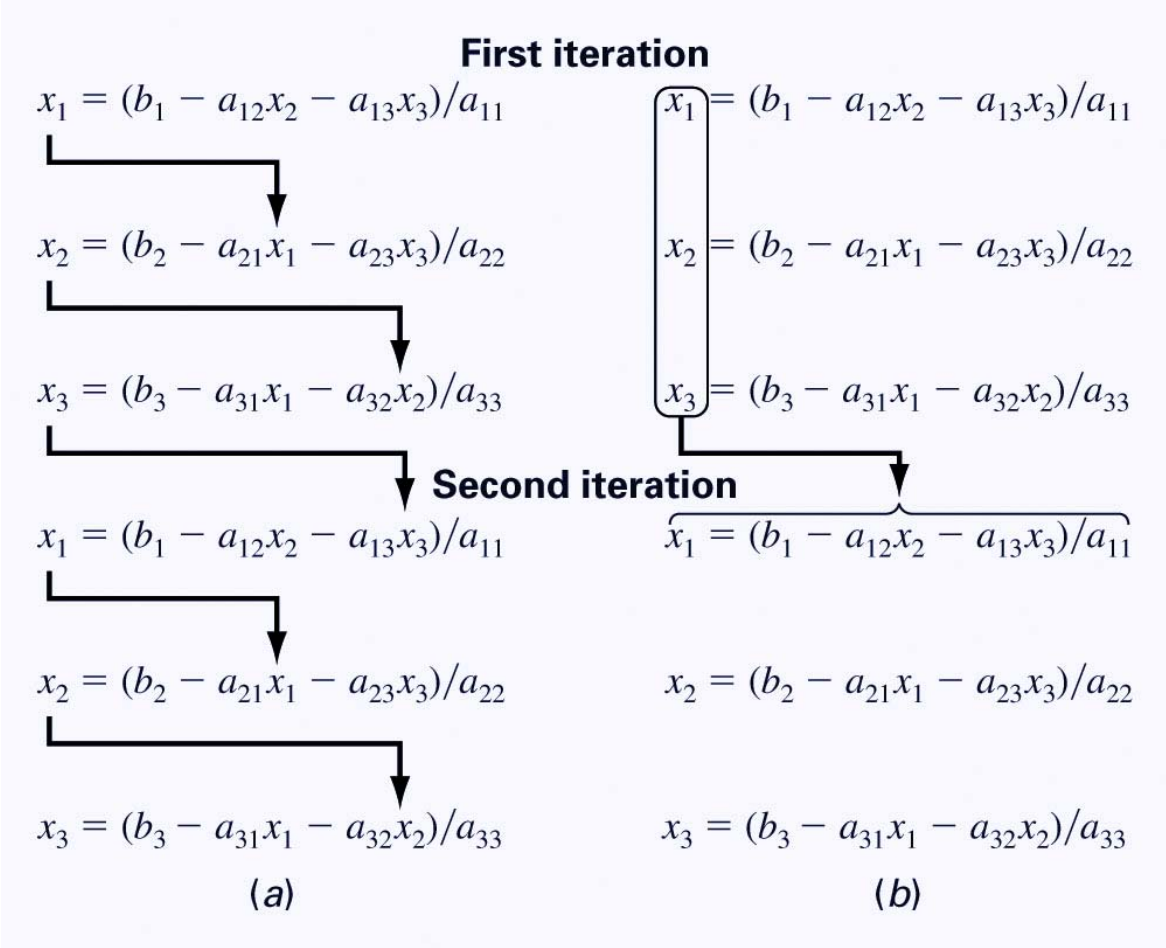
-Note: singular matrix (det A=0) has no inverse

-MATLAB: xc=inv(A)*b

Iterative methods

- Uses an initial guess and refine the guess at each step, converging to the solution vector.

Gauss Seidel and Jacobi method: Graphical depiction



Solving With MATLAB

- MATLAB provides two direct ways to solve systems of linear algebraic equations [A]{x}={b}:
  - Left-division
    x = A\b
  - Matrix inversion
    x = inv(A)*b
- Note: The matrix inverse is less efficient than left-division and also only works for square, non-singular systems.

Sparse matrix computations

$$AS = \begin{bmatrix}
3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
-1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\
0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\
0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 3 & -1 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\
\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3
\end{bmatrix}$$

- No row of A has more than 4 non-zero entries.
- Since fewer than 4n of the n^2 potential entries are non zero.

We may call this matrix Sprase.

Sparse matrix computations

We want to solve a system of equations defined by **AS** for $n=10^5$

What are the options?

Treating the coefficient matrix A as a full matrix means

$\Rightarrow$ storing $n^2=10^{10}$ elements each as a double precision floating point number.

One double precision floating point number requires 8 bytes (64 bit) of storage.

So, $n^2=10^{10}$ elements will require $8*10^{10}$ bytes

= 80 gigabytes (approx.) of RAM

Not only the size is enemy, but so is time.

The number of flops by Gauss Elimination will be order of $n^3 \approx 10^{15}$

If a PC runs on the order of a few GHz (10^9 cycle per second), an upper bound of floating point operations per second is around 10^8.

Therefore, time required of Gauss Elimination=$10^{15}/10^8=10^7$ seconds.

Note: 1 year =$3*10^7$ seconds.

Sparse matrix computations

- So, it is clear that Gauss Elimination method for this problem is not an overnight computation.

On the other hand,

- One step of an iterative method will require $2*4n=8*10^5$ (approx.) operations.
- If the solution needs 100 iterations in Jacobi method, total operations=10^8
- Time required in modern PC for Jacobi method will be

=1 second (approx.) or less !!!