MT308 Numerical Analysis

Girne American University
Faculty of Engineering
Spring 2026
Dr. Kamil Dimililer

1

Gaussian Elimination

Gaussian elimination is an algorithm used to *solve systems of linear equations*, *find the rank of a matrix*, or *compute the inverse of an invertible matrix*.

It transforms a matrix into **row echelon form** (REF) using elementary row operations based on the followings:

1) Swap one equation for another.
2) Add or subtract a multiple of one equation from another.
3) Multiply an equation by a nonzero constant.

2

Steps of the Gaussian Elimination Algorithm:

(Represent the systems of equations as an augmented matrix)

Forward Elimination: Transform the matrix into upper triangular (row echelon) form:

1. **Pivot Selection**: Start at the top-left element. If it's zero, swap with a row below where the element in the same column is non-zero.
2. **Normalize Pivot Row**: Make the pivot = 1 by dividing the entire row.
3. **Eliminate Below**: Subtract multiples of the pivot row from the rows below to make all entries below the pivot zero.
4. **Move to the Next Column and Repeat** for all pivot positions.

Back Substitution: Once the matrix is in upper triangular form:

1. **Solve from the bottom row up**.
2. **Substitute the known values** back into the equations above to find all variables.

3

Example: Solve the following system using the Gaussian Elimination method

$$2x-3y+z=-10$$

$$5x+2y+3z=-17$$

$$-3x+4y-2z=16$$

Solution: Write the problem in Ax=b form:

$$\begin{pmatrix} 2 & -3 & 1 \\ 5 & 2 & 3 \\ -3 & 4 & -2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -10 \\ -17 \\ 16 \end{pmatrix}$$

The Augmented matrix: $\begin{pmatrix} 2 & -3 & 1 & -10 \\ 5 & 2 & 3 & -17 \\ -3 & 4 & -2 & 16 \end{pmatrix}$ Normalize the first row using the first element ➡ $\begin{pmatrix} 1.0 & -1.5 & 0.5 & -5.0 \\ 5.0 & 2.0 & 3.0 & -17.0 \\ -3.0 & 4.0 & -2.0 & 16.0 \end{pmatrix}$ Use the first element of the first row as the pivot ➡

➡ $\begin{pmatrix} 1.0 & -1.5 & 0.5 & -5.0 \\ 0 & 9.5 & 0.5 & 8.0 \\ 0 & -0.5 & -0.5 & 1.0 \end{pmatrix}$ Normalize the second row using the its second element ➡ $\begin{pmatrix} 1 & -1.5 & 0.5 & -5.0 \\ 0 & 1.0 & 0.0526 & 0.8421 \\ 0 & -0.5 & -0.5 & 1.0 \end{pmatrix}$ Use the second element of the second row as the pivot ➡ $\begin{pmatrix} 1.0 & -1.5 & 0.5000 & -5.000 \\ 0 & 1 & 0.0526 & 0.8421 \\ 0 & 0 & -0.4737 & 1.4211 \end{pmatrix}$

4

Normalize the third row using its third element

$$\begin{bmatrix} 1 & -1.5 & 0.5000 & -5.0000 \\ 0 & 1.0 & 0.0526 & 0.8421 \\ 0 & 0 & 1.0000 & -3.0000 \end{bmatrix}$$

Apply the back substitution to find z, y and x

z=-3

y=0.8421-0.0526 z=-1

x= -5 +1.5y-0.5z=-2

The following code produces all steps shown in this solution:

```
A=[2  -3  1;5  2  3;-3  4  -2]
b=[-10 -17 16]'
Ab=[A,b]
Ab(1,:)=Ab(1,:)/2
Ab(2,:)=Ab(2,:)-5*Ab(1,:)
Ab(3,:)=Ab(3,:)+3*Ab(1,:)
Ab(2,:)=Ab(2,:)/9.5
Ab(3,:)=Ab(3,:)+0.5*Ab(2,:)
Ab(3,:)=Ab(3,:)/Ab(3,3)
z=Ab(3,end)
y=Ab(2,end)-Ab(2,3)*z
x=Ab(1,end)-Ab(1,2)*y-
Ab(1,3)*z
```

5

A very simple code to implement Gaussian elimination method

```
function x = my_gauss_elimination(A, b)
    [n, m] = size(A);
    if n ~= m
        error('Matrix A must be square.');
    end
    if length(b) ~= n
        error('Vector b must have the same number of rows as A.');
    end

    Ab = [A b];

    for i = 1:n
        for j = i+1:n
            factor = Ab(j,i) / Ab(i,i);
            Ab(j,:) = Ab(j,:) - factor * Ab(i,:);
        end
    end

    x = zeros(n, 1);
    for i = n:-1:1
        x(i) = (Ab(i,end) - Ab(i,i+1:n) * x(i+1:n)) / Ab(i,i);
    end
end
```

Normalization is not compulsory
What happens to the element Ab(i ,i) is zero?

6

In order to avoid division by zero error and to decrease the round-off error of the algorithm, the row with the highest element at the i^{th} column is moved to the i^{th} row:

```
[~, maxRow] = max(abs(Ab(i:n, i)));
maxRow = maxRow + i - 1;
if maxRow ~= i
    Ab([i, maxRow], :) = Ab([maxRow, i], :);
end
```

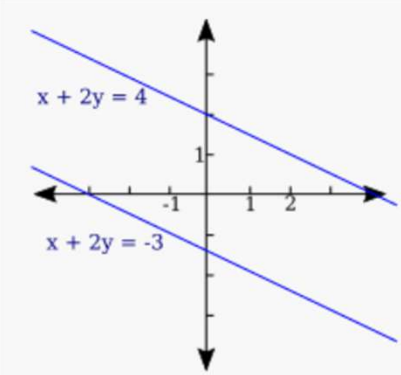
Gaussian elimination method has a computational complexity of $O(n^3)$.

For iterative methods, the number of scalar multiplications is $O(n^2)$ at each iteration.

7

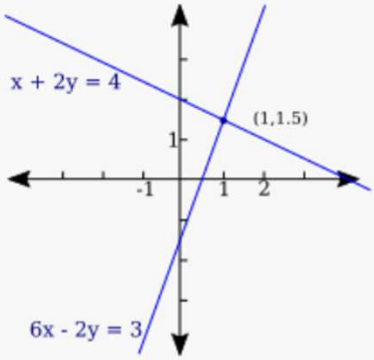
```
% Source: https://www.mathworks.com/matlabcentral/answers/340236-gaussian-elimination-technique-by-matlab
% An alternative Matlab Program to solve A system of equations
clear ; clc ; close all
n = input('Please Enter the size of the equation system n = ') ;
C = input('Please Enter the elements of the Matrix C ' ) ;
b = input('Please Enter the elements of the Matrix b ' ) ;
dett = det(C)
if dett == 0
  print('This system unsolvable because det(C) = 0 ')
else
  b = b'
  A = [ C b ]
for j = 1:(n-1)
  for i= (j+1) : n
      mult = A(i,j)/A(j,j) ;
      for k= j:n+1
          A(i,k) = A(i,k) - mult*A(j,k) ;
          A
      end
  end
end
for p = n:-1:1
  for r = p+1:n
      x(p) = A(p,r)/A(p,r-1)
  end
end
end %end of else
```

8



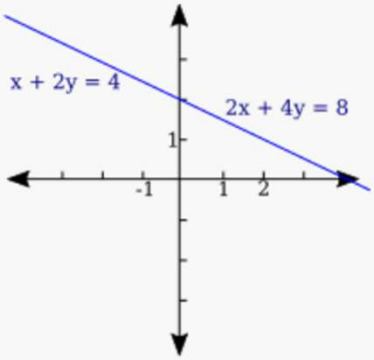
Parallel lines, no intersection.

Echelon form:
x + 2y = 4
0 = -7
No solution.



Two lines intersect at a point.

Echelon form:
x + 2y = 4
-14y = -21
No free variables,
unique solution.



Two lines actually the same line

Echelon form:
x + 2y = 4
0 = 0
One free variable,
infinite solutions..

9

When we apply the Gaussian elimination to a singular system, there are two possible cases:

1. A row of all zeros appears: $0x + 0y + 0z = 0$

   This means some of the equations are not linearly independent. Therefore, there is no unique solution.

2. A contradictory row may appear: $0x + 0y + 0z = 5$

   This means the system is inconsistent, and no solution exists

10

Complexity The operation count shows that direct solution of n equations in n unknowns by Gaussian elimination is an $O(n^3)$ process. This is a useful fact for estimating time required for solving large systems. For example, to estimate the time needed to solve a system of $n = 500$ equations on a particular computer, we could get a fair guess by solving a system of $n = 50$ equations and then scaling the elapsed time by $10^3 = 1000$.

Exercise Questions

1. Use Gaussian elimination to solve the systems:

$$\text{(a)}\ \begin{aligned} 2x - 3y &= 2 \\ 5x - 6y &= 8 \end{aligned} \qquad \text{(b)}\ \begin{aligned} x + 2y &= -1 \\ 2x + 3y &= 1 \end{aligned} \qquad \text{(c)}\ \begin{aligned} -x + y &= 2 \\ 3x + 4y &= 15 \end{aligned}$$

2. Use Gaussian elimination to solve the systems:

$$\text{(a)}\ \begin{aligned} 2x - 2y - z &= -2 \\ 4x + y - 2z &= 1 \\ -2x + y - z &= -3 \end{aligned} \qquad \text{(b)}\ \begin{aligned} x + 2y - z &= 2 \\ 3y + z &= 4 \\ 2x - y + z &= 2 \end{aligned} \qquad \text{(c)}\ \begin{aligned} 2x + y - 4z &= -7 \\ x - y + z &= -2 \\ -x + 3y - 2z &= 6 \end{aligned}$$

11

Iterative Methods to Solve Linear Systems

- Iterative methods are useful if the size of matrix is huge and if there are many 0 elements. During the iterations matrix is not changed so sparcity is preserved
- Starts with an initial approximation for the solution vector (x^0)
- At each iteration updates the x vector by using the sytem Ax=b
- Each iteration involves a matrix-vector product
- Write the system Ax=b in an equivalent form **x=Ex+f**
- (like x=g(x) for fixed-point iteration)
- Starting with x^0, generate a sequence of approximations $\{x^k\}$ iteratively by **x^{k+1}=Ex^k+f**

12

12

Jacobi's Method

The Jacobi iterative method is obtained by solving the ith equation in Ax = b for x_i to obtain (provided $a_{ii} \neq 0$)

$$x_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \left(-\frac{a_{ij}x_j}{a_{ii}} \right) + \frac{b_i}{a_{ii}}, \qquad \text{for } i = 1, 2, \ldots, n.$$

For each k ≥ 1, generate the components $x_i^{(k)}$ of $x^{(k)}$ from the components of $x^{(k-1)}$ by

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[\sum_{\substack{j=1 \\ j \neq i}}^{n} \left(-a_{ij}x_j^{(k-1)} \right) + b_i \right], \qquad \text{for } i = 1, 2, \ldots, n.$$

13

Apply the Jacobi Method to the system $3u + v = 5, u + 2v = 5$.

$$u = \frac{5 - v}{3}$$
$$v = \frac{5 - u}{2}.$$

use the initial guess $(u_0, v_0) = (0, 0)$

The two equations are iterated:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-5/2}{3} \\ \frac{5-5/3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{6} \\ \frac{5}{3} \end{bmatrix}$$

$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} \frac{5-5/3}{3} \\ \frac{5-5/6}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{25}{12} \end{bmatrix}.$$

The Jacobi method converges toward the solution [1,2].

14

Apply the Jacobi Method to the system $u + 2v = 5, 3u + v = 5$.

$$u = 5 - 2v$$
$$v = 5 - 3u.$$

use the initial guess $(u_0, v_0) = (0, 0)$

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 5 - 2v_0 \\ 5 - 3u_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$
$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} 5 - 2v_1 \\ 5 - 3u_1 \end{bmatrix} = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$$
$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} 5 - 2(-10) \\ 5 - 3(-5) \end{bmatrix} = \begin{bmatrix} 25 \\ 20 \end{bmatrix}$$

Diverges

15

The Jacobi iteration method for solving a linear system converges if any of the following conditions is satisfied:

1. A **is strictly diagonally dominant**
   For every row i:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

   This is a *sufficient* condition, not necessary.

2. A **is symmetric positive definite (SPD)**
   If A is symmetric ($A = A^T$) and $x^T Ax > 0$ for all $x \neq 0$, Jacobi converges.
   (However, Gauss-Seidel generally performs better on SPD systems.)

3. **The spectral radius** $\rho(G_J) < 1$
   Where $G_J = D^{-1}(L + U)$ is the Jacobi iteration matrix.
   This is the **necessary and sufficient condition for convergence**.
   $\rho(G_J)$ is the largest absolute value of the eigenvalues of G_J.

16

```
function x = jacobi_it(A,b,x0,tol)
n = length(b);
x = x0;
while norm(A*x - b, inf) > tol
    x_new = zeros(n,1);
    for i = 1:n
        s = A(i,1:i-1)*x(1:i-1) + A(i,i+1:n)*x(i+1:n);
        x_new(i) = (b(i) - s)/A(i,i);
    end
    x = x_new;
end
end
```

17