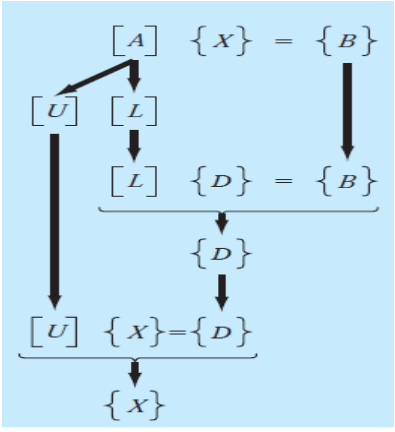
Lecture 10: *LU* Decomposition & Matrix Inversion



Introduction

- *LU* decomposition techniques are a class of elimination methods.
- The primary appeal of *LU* decomposition is that the time-consuming elimination step can be formulated so that it involves only operations on the matrix of coefficients, [*A*].
- Thus, it is well suited for those situations where many right-hand-side vectors {*B*} must be evaluated for a single value of [*A*].
- Although there are a variety of ways in which this is done, we will focus on showing how the Gauss elimination method can be implemented as an *LU* decomposition.
- One motive for introducing *LU* decomposition is that it provides an efficient means to compute the matrix inverse.
- The inverse has a number of valuable applications in engineering practice.
- It also provides a means for evaluating system condition.

LU Decomposition

$$[A]\{X\} = \{B\}$$

- Recall that Gauss elimination involves two steps: forward elimination and back substitution
- Of these, the forward-elimination step comprises the bulk of the computational effort .
- This is particularly true for large systems of equations.
- *LU* decomposition methods separate the time consuming elimination of the matrix [*A*] from the manipulations of the right-hand side {*B*}.
- Thus, once [*A*] has been "decomposed," multiple right-hand-side vectors can be evaluated in an efficient manner.
- Interestingly, Gauss elimination itself can be expressed as an *LU* decomposition.

Overview of *LU* Decomposition

- Just as was the case with Gauss elimination, *LU* decomposition requires pivoting to avoid division by zero.
- The following explanation is limited to a set of three simultaneous equations. The results can be directly extended to *n*-dimensional systems.

$$[A]\{X\} = \{B\} \qquad [A]\{X\} - \{B\} = 0$$

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} \qquad [U]\{X\} - \{D\} = 0$$

Overview of *LU* Decomposition

$$[U]\{X\} - \{D\} = 0 \qquad [L] = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}$$

- Now, assume that there is a lower diagonal matrix with 1's on the diagonal, that has the property that when the equation above is premultiplied by it, equation below is the result.

$$[A]\{X\} - \{B\} = 0$$

$$[L]\{[U]\{X\} - \{D\}\} = [A]\{X\} - \{B\}$$

Overview of *LU* Decomposition

$$[L]\{[U]\{X\} - \{D\}\} = [A]\{X\} - \{B\}$$

- If this equation holds, it follows from the rules for matrix multiplication that

$$[L][U] = [A] \qquad [L]\{D\} = \{B\}$$

Overview of *LU* Decomposition

$$[U]\{X\} - \{D\} = 0 \qquad [L][U] = [A] \qquad [L]\{D\} = \{B\}$$

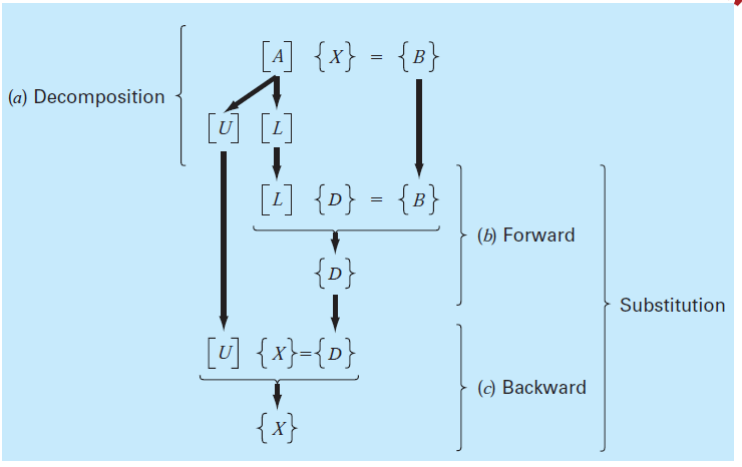
- A two-step strategy for obtaining solutions can be based on the above three equations.
  1. *LU* decomposition step. [*A*] is factored or "decomposed" into lower [*L*] and upper [*U*] triangular matrices.
  2. *Substitution step.* [*L*] and [*U*] are used to determine a solution {*X*} for a right-hand side {*B*}.

     This step itself consists of two steps:

     First, the **third equation** is used to generate an intermediate vector {*D*} by forward substitution.

     Then, the result is substituted into the **first equation**, which can be solved by back substitution for {*X*}.

The steps in *LU* decomposition



LU Decomposition Version of Gauss Elimination

- Gauss elimination can be used to decompose [*A*] into [*L*] and [*U*].
- This can be easily seen for [*U*], which is a direct product of the forward elimination.

$$[U] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix}$$

LU Decomposition Version of Gauss Elimination

- Though it might not be as apparent, the matrix [*L*] is also produced during the step.
- This can be readily illustrated for a three-equation system,

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

- The first step in Gauss elimination is to multiply row 1 by the factor below and subtract the result from the second row to eliminate a_{21}.

$$f_{21} = \frac{a_{21}}{a_{11}}$$

LU Decomposition Version of Gauss Elimination

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

- Similarly, row 1 is multiplied by the factor below and the result subtracted from the third row to eliminate a_{31}.

$$f_{31} = \frac{a_{31}}{a_{11}}$$

- The final step is to multiply the modified second row by the factor below and subtract the result from the third row to eliminate a'_{32}.

$$f_{32} = \frac{a'_{32}}{a'_{22}}$$

LU Decomposition Version of Gauss Elimination

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

- Now suppose that we merely perform all these manipulations on the matrix [*A*].
- Clearly, if we do not want to change the equation, we also have to do the same to the right hand side {*B*}.
- But there is absolutely no reason that we have to perform the manipulations simultaneously.
- Thus, we could save the *f* 's and manipulate {*B*} later.

LU Decomposition Version of Gauss Elimination

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

- Where do we store the factors f_{21}, f_{31}, and f_{32}?
- Recall that the whole idea behind the elimination was to create zeros in a_{21}, a_{31}, and a_{32}.
- Thus, we can store f_{21} in a_{21}, f_{31} in a_{31}, and f_{32} in a_{32}.
- After elimination, the [*A*] matrix can therefore be written as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ f_{21} & a'_{22} & a'_{23} \\ f_{31} & f_{32} & a''_{33} \end{bmatrix}$$

LU Decomposition Version of Gauss Elimination

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ f_{21} & a'_{22} & a'_{23} \\ f_{31} & f_{32} & a''_{33} \end{bmatrix}$$

- This matrix, in fact, represents an efficient storage of the *LU* decomposition of [*A*],

$$[A] \rightarrow [L][U]$$

$$[U] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix} \qquad [L] = \begin{bmatrix} 1 & 0 & 0 \\ f_{21} & 1 & 0 \\ f_{31} & f_{32} & 1 \end{bmatrix}$$

Example 1

Naive Gauss Elimination

Problem Statement. Use Gauss elimination to solve

$$3x_1 - 0.1x_2 - 0.2x_3 = 7.85$$
$$0.1x_1 + 7x_2 - 0.3x_3 = -19.3$$
$$0.3x_1 - 0.2x_2 + 10x_3 = 71.4$$

Carry six significant figures during the computation.

LU Decomposition with Gauss Elimination

Problem Statement. Derive an LU decomposition based on the Gauss elimination performed in Example 9.5.

Example 1

$$[A] = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix}$$

After forward elimination, the following upper triangular matrix was obtained:

$$[U] = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix}$$

$$f_{21} = \frac{0.1}{3} = 0.03333333 \qquad f_{31} = \frac{0.3}{3} = 0.1000000 \qquad f_{32} = \frac{-0.19}{7.00333} = -0.0271300$$

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix}$$

Example 1

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix} \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix}$$

$$[L][U] = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.0999999 & 7 & -0.3 \\ 0.3 & -0.2 & 9.99996 \end{bmatrix} \qquad [A] = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix}$$

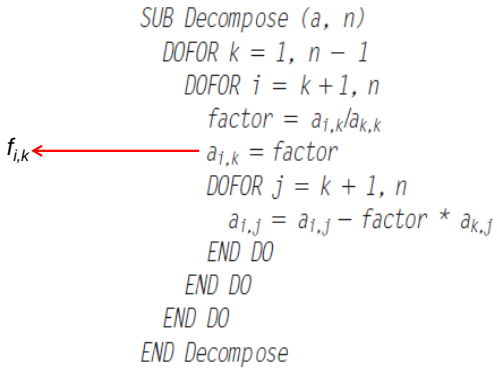
where the minor discrepancies are due to round-off.

Substitution Phases

$$[L]\{D\} = \{B\} \qquad [U]\{X\} - \{D\} = 0$$

- After the matrix is decomposed, a solution can be generated for a particular right hand-side vector {*B*}.
- This is done in two steps.
  - First, a forward-substitution step is executed by solving the first equation above for {*D*}.
    - It is important to recognize that this merely amounts to performing the elimination manipulations on {*B*}.
    - Thus, at the end of this step, the right hand side will be in the same state that it would have been had we performed forward manipulation on [*A*] and {*B*} simultaneously.
  - The second step then merely amounts to implementing back substitution, as in the second equation above.
    - Again, it is important to recognize that this is identical to the back-substitution phase of conventional Gauss elimination.

Pseudocode for Decomposition Phase



- Notice that this algorithm is “naive” in the sense that pivoting is not included.

Substitution Phases

$$[L]\{D\} = \{B\}$$

- Forward-substitution
- The forward-substitution step can be represented concisely as

$$d_i = b_i - \sum_{j=1}^{i-1} a_{ij} d_j \qquad \text{for } i = 2, 3, \ldots, n$$

- Back-substitution
- The Back-substitution step can be represented concisely as

$$x_n = d_n / a_{nn}$$

$$x_i = \frac{d_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}} \qquad \text{for } i = n-1, n-2, \ldots, 1$$

Example 2

The Substitution Steps

Problem Statement. Complete the problem initiated in Example 10.1 by generating the final solution with forward and back substitution.

$$\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.3 \\ 71.4 \end{Bmatrix}$$

$$\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.5617 \\ 70.0843 \end{Bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.3 \\ 71.4 \end{Bmatrix}$$

Example 2

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.3 \\ 71.4 \end{Bmatrix}$$

$$\begin{aligned} d_1 &= 7.85 \\ 0.0333333d_1 + d_2 &= -19.3 \\ 0.1d_1 - 0.02713d_2 + d_3 &= 71.4 \end{aligned}$$

$$d_1 = 7.85$$

$$d_2 = -19.3 - 0.0333333(7.85) = -19.5617$$

$$d_3 = 71.4 - 0.1(7.85) + 0.02713(-19.5617) = 70.0843$$

$$\{D\} = \begin{Bmatrix} 7.85 \\ -19.5617 \\ 70.0843 \end{Bmatrix}$$

Example 2

$$\{D\} = \begin{Bmatrix} 7.85 \\ -19.5617 \\ 70.0843 \end{Bmatrix}$$

$$[U]\{X\} = \{D\}$$

$$\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.5617 \\ 70.0843 \end{Bmatrix}$$

$$\{X\} = \begin{Bmatrix} 3 \\ -2.5 \\ 7.00003 \end{Bmatrix}$$

Pseudocode for Both Substitution Phases

```
SUB Substitute (a, n, b, x)
  'forward substitution
  DOFOR i = 2, n
    sum = bi
    DOFOR j = 1, i - 1
      sum = sum - ai,j * bj
    END DO
    bi = sum
  END DO
  'back substitution
  xn = bn / an,n
  DOFOR i = n - 1, 1, -1
    sum = 0
    DOFOR j = i + 1, n
      sum = sum + ai,j * xj
    END DO
    xi = (bi - sum)/ai,i
  END DO
END Substitute
```

Computational Effort

- The *LU* decomposition algorithm requires the same total multiply/divide flops as for Gauss elimination.
- The only difference is that a little less effort is expended in the decomposition phase since the operations are not applied to the right-hand side.
- Thus, the number of multiply/divide flops involved in the decomposition phase can be calculated as

$$\frac{n^3}{3} - \frac{n}{3} \xrightarrow{\text{as } n \text{ increases}} \frac{n^3}{3} + O(n)$$

  - Conversely, the substitution phase takes a little more effort.
  - Thus, the number of flops for forward and back substitution is n^2.
  - The total effort is therefore identical to Gauss elimination

$$\frac{n^3}{3} - \frac{n}{3} + n^2 \xrightarrow{\text{as } n \text{ increases}} \frac{n^3}{3} + O(n^2)$$

LU Decomposition Algorithm

```
SUB Ludecomp (a, b, n, tol, x, er)
  DIM o(n), s(n)
  er = 0
  CALL Decompose(a, n, tol, o, s, er)
  IF er <> -1 THEN
     CALL Substitute(a, o, n, b, x)
  END IF
END Ludecomp

SUB Decompose (a, n, tol, o, s, er)
  DOFOR i = 1, n
    o(i) = i
    s(i) = ABS(a(i,1))
    DOFOR j = 2, n
       IF ABS(a(i,j))>s(i) THEN s(i) = ABS(a(i,j))
    END DO
  END DO
  DOFOR k = 1, n - 1
    CALL Pivot(a, o, s, n, k)
    IF ABS(a(o(k),k)/s(o(k))) < tol THEN
       er = -1
       PRINT a(o(k),k)/s(o(k))
       EXIT DO
    END IF
    DOFOR i = k + 1, n
       factor = a(o(i),k)/a(o(k),k)
       a(o(i),k) = factor
       DOFOR j = k + 1, n
         a(o(i),j) = a(o(i),j) - factor * a(o(k),j)
       END DO
    END DO
  END DO
  IF ABS(a(o(k),k)/s(o(k))) < tol THEN
     er = -1
     PRINT a(o(k),k)/s(o(k))
  END IF
END Decompose

SUB Pivot (a, o, s, n, k)
  p = k
  big = ABS(a(o(k),k)/s(o(k)))
  DOFOR ii = k + 1, n
    dummy = ABS(a(o(ii),k)/s(o(ii)))
    IF dummy > big THEN
       big = dummy
       p = ii
    END IF
  END DO
  dummy = o(p)
  o(p) = o(k)
  o(k) = dummy
END Pivot

SUB Substitute (a, o, n, b, x)
  DOFOR i = 2, n
    sum = b(o(i))
    DOFOR j = 1, i - 1
      sum = sum - a(o(i),j) * b(o(j))
    END DO
    b(o(i)) = sum
  END DO
  x(n) = b(o(n))/a(o(n),n)
  DOFOR i = n - 1, 1, -1
    sum = 0
    DOFOR j = i + 1, n
      sum = sum + a(o(i),j) * x(j)
    END DO
    x(i) = (b(o(i)) - sum)/a(o(i),i)
  END DO
END Substitute
```

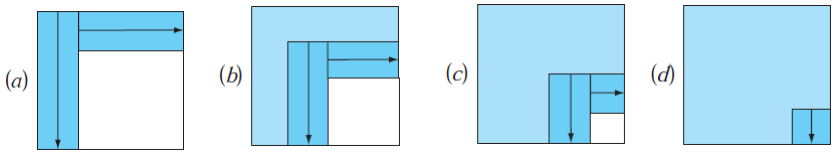
LU Decomposition Algorithm

- An algorithm to implement an *LU* decomposition expression of Gauss elimination is given in the previous slide.
- Four features of this algorithm bear mention:
  1. The factors generated during the elimination phase are stored in the lower part of the matrix → saves storage space.
  2. This algorithm keeps track of pivoting by using an order vector *o.* This greatly speeds up the algorithm because only the order vector (as opposed to the whole row) is pivoted.
  3. The equations are not scaled, but scaled values of the elements are used to determine whether pivoting is to be implemented.
  4. The diagonal term is monitored during the pivoting phase to detect near-zero occurrences in order to flag singular systems.

Crout Decomposition

- Notice that for the *LU* decomposition implementation of Gauss elimination, the [*L*] matrix has 1's on the diagonal.
- This is formally referred to as a Doolittle decomposition, or factorization.
- An alternative approach involves a [*U*] matrix with 1's on the diagonal.
- This is called Crout decomposition.
- Although there are some differences between the approaches (Atkinson, 1978; Ralston and Rabinowitz, 1978), their performance is comparable.

Crout Decomposition



- The Crout decomposition approach generates [*U*] and [*L*] by sweeping through the matrix by columns and rows.
- Concise series of formulas is available for its implementation.

Crout Decomposition

$$l_{i,1} = a_{i,1} \qquad \text{for } i = 1, 2, \ldots, n$$

$$u_{1j} = \frac{a_{1j}}{l_{11}} \qquad \text{for } j = 2, 3, \ldots, n$$

For $j = 2, 3, \ldots, n-1$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \qquad \text{for } i = j, j+1, \ldots, n$$

$$u_{jk} = \frac{a_{jk} - \sum_{i=1}^{j-1} l_{ji} u_{ik}}{l_{jj}} \qquad \text{for } k = j+1, j+2, \ldots, n$$

$$l_{nn} = a_{nn} - \sum_{k=1}^{n-1} l_{nk} u_{kn}$$

Crout Decomposition

- Aside from the fact that it consists of a few concise loops, the foregoing approach also has the benefit that storage space can be economized.
- There is no need to store the 1's on the diagonal of [*U*] or the 0's for [*L*] or [*U*] because they are givens in the method.
- Consequently, the values of [*U*] can be stored in the zero space of [*L*].
- Further, close examination of the foregoing derivation makes it clear that after each element of [*A*] is employed once, it is never used again.
- Therefore, as each element of [*L*] and [*U*] is computed, it can be substituted for the corresponding element (as designated by its subscripts) of [*A*].

The Matrix Inverse

- If a matrix [*A*] is square, there is another matrix, $[A]^{-1}$, called the inverse of [*A*], for which

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

- Our focus:
  1. How to calculate matrix inverse?
  2. How to use it for engineering analysis?

Calculating the Inverse

$$\{b\} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \qquad \{b\} = \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix}$$

- The best way to implement such a calculation is with the *LU* decomposition algorithm.

Example 3

Matrix Inversion

Problem Statement. Employ *LU* decomposition to determine the matrix inverse for the system from Example 10.2.

$$[A] = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix}$$

Recall that the decomposition resulted in the following lower and upper triangular matrices:

$$[U] = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix} \qquad [L] = \begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix}$$

Example 3

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.0333333 & 1 & 0 \\ 0.100000 & -0.0271300 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \qquad \{D\}^T = \lfloor 1 \quad -0.03333 \quad -0.1009 \rfloor$$

$$\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.293333 \\ 0 & 0 & 10.0120 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ -0.03333 \\ -0.1009 \end{Bmatrix} \qquad \{X\}^T = \lfloor 0.33249 \quad -0.00518 \quad -0.01008 \rfloor$$

$$[A]^{-1} = \begin{bmatrix} 0.33249 & 0 & 0 \\ -0.00518 & 0 & 0 \\ -0.01008 & 0 & 0 \end{bmatrix} \qquad [A]^{-1} = \begin{bmatrix} 0.33249 & 0.004944 & 0.006798 \\ -0.00518 & 0.142903 & 0.004183 \\ -0.01008 & 0.00271 & 0.09988 \end{bmatrix}$$

The validity of this result can be checked by verifying that $[A][A]^{-1} = [I]$

Matrix Inverse: Computational Effort

- The effort required for this algorithm is simply computed as

$$\underbrace{\frac{n^3}{3} - \frac{n}{3}}_{\text{decomposition}} + \underbrace{n(n^2)}_{n \times \text{substitutions}} = \frac{4n^3}{3} - \frac{n}{3}$$

Stimulus-Response Computations

- Many of the linear systems of equations confronted in engineering practice are derived from conservation laws.
- The mathematical expression of these laws is some form of balance equation to ensure that a particular property—mass, force, heat, momentum, or other—is conserved.
- A single balance equation can be written for each part of the system, resulting in a set of equations defining the behavior of the property for the entire system.
- These equations are interrelated, or coupled, in that each equation may include one or more of the variables from the other equations.

Stimulus-Response Computations

- For many cases, these systems are linear and, therefore, of the exact form given below:

$$[A]\{X\} = \{B\}$$

- Now, for balance equations, the terms of the above equation have a definite physical interpretation.
- For example, the elements of {*X*} are the levels of the property being balanced for each part of the system.
- In a force balance of a structure, they represent the horizontal and vertical forces in each member.
- They represent the system's state or response, which we are trying to determine.

Stimulus-Response Computations

- The right-hand-side vector {*B*} contains those elements of the balance that are independent of behavior of the system—that is, they are constants.
- As such, they often represent the external forces or stimuli that drive the system.
- Finally, the matrix of coefficients [*A*] usually contains the parameters that express how the parts of the system interact or are coupled.
- Consequently, this equation might be reexpressed as

$$[\text{Interactions}]\{\text{response}\} = \{\text{stimuli}\}$$

Stimulus-Response Computations

$$[A]\{X\} = \{B\}$$

- Therefore, the equation above can be seen as a fundamental mathematical model designed for coupled systems involving several dependent variables {*X*}.
- This system can be solved in several ways including using matrix inverse:

$$\{X\} = [A]^{-1}\{B\}$$

$$\begin{aligned} x_1 &= a_{11}^{-1}b_1 + a_{12}^{-1}b_2 + a_{13}^{-1}b_3 \\ x_2 &= a_{21}^{-1}b_1 + a_{22}^{-1}b_2 + a_{23}^{-1}b_3 \\ x_3 &= a_{31}^{-1}b_1 + a_{32}^{-1}b_2 + a_{33}^{-1}b_3 \end{aligned}$$

Stimulus-Response Computations

$$x_1 = a_{11}^{-1} b_1 + a_{12}^{-1} b_2 + a_{13}^{-1} b_3$$
$$x_2 = a_{21}^{-1} b_1 + a_{22}^{-1} b_2 + a_{23}^{-1} b_3$$
$$x_3 = a_{31}^{-1} b_1 + a_{32}^{-1} b_2 + a_{33}^{-1} b_3$$

- Thus, we find that the inverted matrix itself, aside from providing a solution, has extremely useful properties.
- Each of its elements represents the response of a single part of the system to a unit stimulus of any other part of the system.

Stimulus-Response Computations

$$x_1 = a_{11}^{-1} b_1 + a_{12}^{-1} b_2 + a_{13}^{-1} b_3$$
$$x_2 = a_{21}^{-1} b_1 + a_{22}^{-1} b_2 + a_{23}^{-1} b_3$$
$$x_3 = a_{31}^{-1} b_1 + a_{32}^{-1} b_2 + a_{33}^{-1} b_3$$

- Notice that these formulations are linear and, therefore, superposition and proportionality hold.
- Superposition means that if a system is subject to several different stimuli (the *b*'s), the responses can be computed individually and the results summed to obtain a total response.
- Proportionality means that multiplying the stimuli by a quantity results in the response to those stimuli being multiplied by the same quantity.

Stimulus-Response Computations

$$x_1 = a_{11}^{-1} b_1 + a_{12}^{-1} b_2 + a_{13}^{-1} b_3$$
$$x_2 = a_{21}^{-1} b_1 + a_{22}^{-1} b_2 + a_{23}^{-1} b_3$$
$$x_3 = a_{31}^{-1} b_1 + a_{32}^{-1} b_2 + a_{33}^{-1} b_3$$

- Thus, the coefficient a_{11}^{-1} is a proportionality constant that gives the value of x_1 due to a unit level of b_1.
- This result is independent of the effects of b_2 and b_3 on x_1, which are reflected in the coefficients a_{12}^{-1} and a_{13}^{-1}, respectively.
- Therefore, we can draw the general conclusion that the element a_{ij}^{-1} of the inverted matrix represents the value of x_i due to a unit quantity of b_j.

Stimulus-Response Computations

$$x_1 = a_{11}^{-1} b_1 + a_{12}^{-1} b_2 + a_{13}^{-1} b_3$$
$$x_2 = a_{21}^{-1} b_1 + a_{22}^{-1} b_2 + a_{23}^{-1} b_3$$
$$x_3 = a_{31}^{-1} b_1 + a_{32}^{-1} b_2 + a_{33}^{-1} b_3$$

- Using the example of the structure, element a_{ij}^{-1} of the matrix inverse would represent the force in member *i* due to a unit external force at node *j*.
- Even for small systems, such behavior of individual stimulus-response interactions would not be intuitively obvious.
- As such, the matrix inverse provides a powerful technique for understanding the interrelationships of component parts of complicated systems.

Error Analysis and System Condition

- Matrix inverse also provides a means to discern whether systems are ill-conditioned.
  1. Scale the matrix of coefficients [*A*] so that the largest element in each row is 1. Invert the scaled matrix and if there are elements of $[A]^{-1}$ that are several orders of magnitude greater than one, it is likely that the system is ill-conditioned.
  2. Multiply the inverse by the original coefficient matrix and assess whether the result is close to the identity matrix. If not, it indicates ill-conditioning.
  3. Invert the inverted matrix and assess whether the result is sufficiently close to the original coefficient matrix. If not, it again indicates that the system is ill-conditioned.

Interpreting Elements of Matrix Inverse as a Measure of Ill-Conditioning

$$\{R\} = \{B\} - [A]\{\tilde{X}\}$$

- Where is ˜*X* an approximate solution.
- Suppose that {*X*} is the exact solution that yields a zero residual.

$$\{0\} = \{B\} - [A]\{X\} \qquad \{R\} = [A]\left\{\{X\} - \{\tilde{X}\}\right\}$$

$$\{X\} - \{\tilde{X}\} = [A]^{-1}\{R\}$$

Interpreting Elements of Matrix Inverse as a Measure of Ill-Conditioning

$$\{X\} - \{\tilde{X}\} = [A]^{-1}\{R\}$$

- This result indicates why checking a solution by substitution can be misleading.
- For cases where elements of $[A]^{-1}$ are large, a small discrepancy in the right-hand-side residual $\{R\}$ could correspond to a large error $\{X\} - \{\tilde{X}\}$ in the calculated value of the unknowns.
- In other words, a small residual does not guarantee an accurate solution.
- However, we can conclude that if the largest element of $[A]^{-1}$ is on the order of magnitude of unity, the system can be considered to be well-conditioned.
- Conversely, if $[A]^{-1}$ includes elements much larger than unity, we conclude that the system is ill-conditioned.

Vector and Matrix Norms

- It would be preferable to obtain a single number (such as the condition number) that could serve as an indicator of the problem of ill-conditioning.
- Attempts to formulate such a matrix condition number are based on the mathematical concept of the norm.

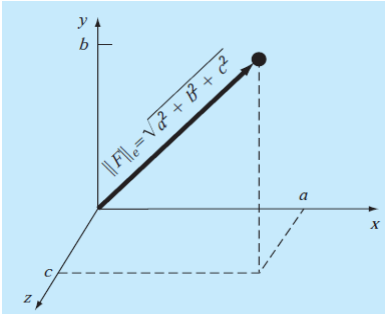
Vector and Matrix Norms

- A norm is a real-valued function that provides a measure of the size or "length" of multi-component mathematical entities such as vectors and matrices.
- A simple example is a vector in three-dimensional Euclidean space that can be represented as

$$\lfloor F \rfloor = \lfloor a \quad b \quad c \rfloor$$

$$\| F \|_e = \sqrt{a^2 + b^2 + c^2}$$

Euclidean norm



Vector and Matrix Norms

- Similarly, for an *n*-dimensional vector $X = [x_1 \quad x_2 \cdots x_n]$, a Euclidean norm would be computed as

$$\| X \|_e = \sqrt{\sum_{i=1}^{n} x_i^2}$$

- The concept can be extended further to a matrix [*A*], as in

$$\| A \|_e = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^2}$$

Special name—the Frobenius norm

Vector and Matrix Norms

- There are alternatives to the Euclidean and Frobenius norms.
- For example, a uniform vector norm is defined as

$$\|X\|_\infty = \max_{1\le i\le n} |x_i|$$

- Similarly, a uniform matrix norm or row-sum norm is defined as

$$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|$$

Matrix Condition Number

$$\text{Cond}\,[A] = \|A\| \cdot \|A^{-1}\|$$

- Note that for a matrix [*A*], this number will be greater than or equal to 1.
- It can be shown (Ralston and Rabinowitz, 1978; Gerald and Wheatley, 1989) that

$$\frac{\|\Delta X\|}{\|X\|} \le \text{Cond}\,[A]\frac{\|\Delta A\|}{\|A\|}$$

- The relative error of the norm of the computed solution can be as large as the relative error of the norm of the coefficients of [*A*] multiplied by the condition number.

Matrix Condition Number

$$\frac{\|\Delta X\|}{\|X\|} \leq \text{Cond}\,[A]\frac{\|\Delta A\|}{\|A\|}$$

- For example, if the coefficients of [*A*] are known to *t*-digit precision (that is, rounding errors are on the order of 10^{-t}) and Cond [A] = 10^{c}, the solution [*X*] may be valid to only *(t−c)* digits (rounding errors $\sim 10^{c-t}$).

Example 4

Matrix Condition Evaluation

Problem Statement. The Hilbert matrix, which is notoriously ill-conditioned, can be represented generally as

$$\begin{bmatrix} 1 & 1/2 & 1/3 & \cdots & 1/n \\ 1/2 & 1/3 & 1/4 & \cdots & 1/(n+1) \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ 1/n & 1/(n+1) & 1/(n+2) & \cdots & 1/(2n-1) \end{bmatrix}$$

Use the row-sum norm to estimate the matrix condition number for the 3×3 Hilbert matrix,

$$[A] = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$$

Example 4

Solution. First, the matrix can be normalized so that the maximum element in each row is 1,

$$[A] = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1 & 2/3 & 1/2 \\ 1 & 3/4 & 3/5 \end{bmatrix} \qquad \|A\|_\infty = 1 + \frac{3}{4} + \frac{3}{5} = 2.35$$

The inverse of the scaled matrix can be computed as

$$[A]^{-1} = \begin{bmatrix} 9 & -18 & 10 \\ -36 & 96 & -60 \\ 30 & -90 & 60 \end{bmatrix}$$

Note that the elements of this matrix are larger than the original matrix. This is also reflected in its row-sum norm, which is computed as

$$\|A^{-1}\|_\infty = |-36| + |96| + |-60| = 192$$

Thus, the condition number can be calculated as

$$\text{Cond}\,[A] = 2.35(192) = 451.2$$

Example 4

- The fact that the condition number is considerably greater than unity suggests that the system is ill-conditioned.
- The extent of the ill-conditioning can be quantified by calculating $c = \log 451.2 = 2.65$.
- Computers using IEEE floating-point representation have approximately $t = \log 2^{-24} = 7.2$ significant base-10 digits.
- Therefore, the solution could exhibit rounding errors of up to $10^{(2.65-7.2)} = 3 \times 10^{-5}$.
- Note that such estimates almost always overpredict the actual error.
- However, they are useful in alerting you to the possibility that round-off errors may be significant.

Assignment-10

- Problems 10.2, 10.7, 10.15, 10.17.