Solving Simultaneous Linear Algebraic Equations

Larry Caretto
Mechanical Engineering 309

Numerical Analysis of Engineering Systems

February 24-26, 2014

California State University Northridge

Outline

- Eigenvalues, eigenvectors and MATLAB
- Review matrix form of equations and Gaussian elimination solution process
- Give details for Gaussian elimination
- Determining non-unique solutions and accuracy considerations
- Other methods
- MATLAB solvers for simultaneous linear equations

Why Eigenvalues/Eigenvectors

- In electrical and mechanical networks, provides fundamental frequencies
- Shows coordinate transformations appropriate for physical problems
- Provides way to express network problem as diagonal matrix
- Transformations based on eigenvectors used in some solutions of **Ax** = **b**

Eigenvalues and Eigenvectors

- Basic definition (**A** square)**: Ax** = λ**x**
- **x** is eigenvector, λ is eigenvalue
- Basic idea is that eigenvector is special vector of matrix **A**; multiplication of **x** by **A** produces **x** multiplied by a constant
- **Ax** = λ**x** => **Ax** – λ**x** = [**A** – **I**λ]**x** = **0**
- Homogenous equations; requires Det [**A** – **I**λ] = 0 for solution other than **x** = **0**

Det[**A** – **I**λ] = 0

$$Det[\mathbf{A}-\mathbf{I}\lambda] = Det\begin{bmatrix} a_{11}-\lambda & a_{12} & a_{13} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22}-\lambda & a_{23} & \cdots & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33}-\lambda & \cdots & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & \cdots & a_{nn}-\lambda \end{bmatrix} = 0$$

- Det[**A** – **I**λ] = 0 produces an n^{th} order equation that has n roots for λ. May have duplicate roots for eigenvalues.

Two-by-two Matrix Eigenvalues

- Quadratic equation with two roots for eigenvalues

$$\begin{vmatrix} a_{11}-\lambda & a_{12} \\ a_{21} & a_{22}-\lambda \end{vmatrix} = (a_{11}-\lambda)(a_{22}-\lambda)-a_{21}a_{12} =$$

$$\lambda^2-(a_{11}+a_{22})\lambda+a_{11}a_{22}-a_{21}a_{12}=0$$

- Eigenvalue solutions

$$\lambda = \frac{(a_{11}+a_{22}) \pm \sqrt{(a_{11}+a_{22})^2-4(a_{11}a_{22}-a_{21}a_{12})}}{2}$$

Det **A**

Two-by-two Matrix Eigenvectors

- Two eigenvectors: $\mathbf{x}_{(1)} = [x_{(1)1}\ x_{(1)2}]^T$ and $\mathbf{x}_{(2)} = [x_{(2)1}\ x_{(2)2}]^T$ $(\mathbf{x}_{(j)} = [x_{(j)1}\ x_{(j)2}]^T)$
- Substitute each eigenvalue solution, λ_j, into $(\mathbf{A} - \mathbf{I}\lambda)\mathbf{x} = \mathbf{0}$ to find all $\mathbf{x}_{(j)}$ components

$$(a_{11} - \lambda_j)x_{(j)1} + a_{12}x_{(j)2} = 0$$
$$a_{21}x_{(j)1} + (a_{22} - \lambda_j)x_{(j)2} = 0$$

Notation: y_i is component i of vector **y**; $\mathbf{z}_{(k)}$ is one of a vector set with components $z_{(k)i}$

California State University Northridge 7

Two-by-two Eigenvectors II

- Eigenvector equations are homogeneous, so eigenvectors are determined only within a multiplicative constant

$$(a_{11} - \lambda_j)x_{(j)1} + a_{12}x_{(j)2} = 0$$
$$a_{21}x_{(j)1} + (a_{22} - \lambda_j)x_{(j)2} = 0$$

- Pick $x_{(j)1} = \alpha$ (arbitrary factor)
- Find $x_{(j)2}$ from either equation

$$x_{(j)2} = \frac{(\lambda_j - a_{11})}{a_{12}}\alpha = \frac{a_{21}}{(\lambda_j - a_{22})}\alpha$$

- Result is same; setting two expressions for $x_{(j)2}$ equal gives equation for λ

$$(a_{11} - \lambda)(a_{22} - \lambda) - a_{21}a_{12} = 0$$

California State University Northridge 8

Two-by-Two Matrix Results

- Have two eigenvalues, λ_1 and λ_2

$$\lambda_1, \lambda_2 = \frac{(a_{11} + a_{22}) \pm \sqrt{(a_{11} + a_{22})^2 - 4(a_{11}a_{22} - a_{21}a_{12})}}{2}$$

- Have two eigenvectors, $\mathbf{x}_{(1)}$ and $\mathbf{x}_{(2)}$

$$\mathbf{x}_{(1)} = \alpha\begin{bmatrix} 1 \\ \frac{(\lambda_1 - a_{11})}{a_{12}} \end{bmatrix} \quad \mathbf{x}_{(2)} = \beta\begin{bmatrix} 1 \\ \frac{(\lambda_2 - a_{11})}{a_{12}} \end{bmatrix}$$

- Can use any values for α and β

California State University Northridge 9

How Many Eigenvalues?

- An n x n matrix has $k \le n$ distinct eigenvalues
- Algebraic multiplicity of an eigenvalue, M_λ, is the number of roots of $\mathrm{Det}[\mathbf{A} - \mathbf{I}\lambda] = 0$ that have the same root, λ
- Geometric multiplicity, m_λ, of eigenvalue is number of linearly independent eigenvectors for this λ

California State University Northridge 10

Diagonalize a Matrix

- For an n x n matrix it is possible to create a matrix, **X**, where each column is one eigenvector
- One can then show that $\mathbf{X}^{-1}\mathbf{A}\mathbf{X} = \mathbf{\Lambda}$, where $\mathbf{\Lambda}$ is a diagonal matrix whose components are the eigenvalues
  - Have MATLAB commands to do this

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_{(1)1} & \boldsymbol{x}_{(2)1} & \cdots & \boldsymbol{x}_{(N)1} \\ \boldsymbol{x}_{(1)2} & \boldsymbol{x}_{(2)2} & \cdots & \boldsymbol{x}_{(N)2} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_{(1)N} & \boldsymbol{x}_{(2)N} & \vdots & \boldsymbol{x}_{(N)N} \end{bmatrix} \quad \boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{\lambda}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\lambda}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \vdots & \boldsymbol{\lambda}_N \end{bmatrix}$$

California State University Northridge 11

More on MATLAB

- The `eig` function of MATLAB can be used to compute eigenvalues and eigenvectors
  - E = eig(A) produces a column vector, **E**, whose elements are the eigenvalues of **A**
  - [V D] = eig(A) produces a square matrix, **V**, whose columns are the eigenvectors of A, and a diagonal matrix, **D**, whose diagonal elements are the eigenvalues
  - Matrix analysis shows that $\mathbf{V}^{-1}\mathbf{A}\mathbf{V} = \mathbf{D}$

California State University Northridge 12

Eigenvalue/Eigenvector

```
>> A
A =  3     7    -2
    -4    14    12
     8     7     0
>> [V D] = eig(A)
V =0.3046 + 0.5278i   0.3046 - 0.5278i   0.3141
  -0.4360 - 0.0344i  -0.4360 + 0.0344i   0.8389
   0.6613             0.6613             0.4446
D =-0.9308 + 6.0205i        0                  0
         0             -0.9308 - 6.0205i       0
         0                   0            18.8616
>> inv(V)*A*V-D
ans =  1.0e-014 *
-0.0222-0.0888i  -0.5759-0.0666i   0.1332+0.3997i
-0.4871+0.0666i  -0.0222+0.0888i   0.0666-0.4441i
 0.6227-0.1445i   0.6208+0.0553i   0     +0.0865i
```


Matrix and Vector Norms

- Generalization of the size of a vector
- A particle with **V** = [1 2 –3] has a speed of sqrt($1^2 + 2^2 + (-3)^2$) = 3.7417
- Different vector norms
  - Usual norm used above is called “2” norm because it uses a power of two
  - One norm is sum of absolute values; in example it is |1| + |2| + |–3| = 6
  - Infinity norm is maximum absolute value
    - Infinity norm = 3 in above example

MATLAB Matrix Norms

```
A =  3     7    -2
    -4    14    12
     8     7   -10
>> B = 2*A;
>> norm(A,2)
ans = 19.4292
>> norm(B,2)
ans = 38.8584
>> norm(A,1)
ans = 28
>> norm(B,1)
ans = 56
>> norm(A,inf)
ans = 30
>> norm(B,inf)
ans = 60
>> norm(A,'fro')
ans = 25.1197
>> norm(B,'fro')
ans = 50.2394
```

Norms for B are twice those for A

$\|A\|_2$ is largest eigenvalue

$$\|A\|_1 = \max_j \sum_{i=1}^{n} |m_{ij}|$$

$$\|A\|_\infty = \max_i \sum_{j=1}^{n} |m_{ij}|$$

$$\|A\|_{Fro} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |m_{ij}|^2}$$

Review Gauss Elimination

- Practical tool for obtaining solutions
- Analytical tool for determining linear dependence or independence
- Basic idea is to manipulate the equations (or data) to make them easier to solve without changing the results
- Systematically create zeros in lower left part of the equations (or data)

Review Upper Triangular Form

- Convert original set of equations to

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \cdots & \cdots & \alpha_{1n-1} & \alpha_{1n} \\ 0 & \alpha_{22} & \alpha_{23} & \cdots & \cdots & \alpha_{2n-1} & \alpha_{2n} \\ 0 & 0 & \alpha_{33} & \cdots & \cdots & \alpha_{3n-1} & \alpha_{3n} \\ \vdots & \vdots & \vdots & \ddots & & & \\ \vdots & \vdots & \vdots & & \ddots & & \\ 0 & 0 & 0 & \cdots & \cdots & \alpha_{n-1n-1} & \alpha_{n-1n} \\ 0 & 0 & 0 & \cdots & \cdots & 0 & \alpha_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix}$$

Review Back Substitution

- Upper triangular form on previous slide is easily solved by back substitution
- $x_n = \beta_n/\alpha_{nn}$
- $x_{n-1} = (\beta_{n-1} - \alpha_{n-1\,n} x_n)/\alpha_{n-1\,n-1}$, *et cetera*
- General equation for back substitution

$$x_i = \frac{\beta_i - \sum_{j=i+1}^{n} \alpha_{ij} x_j}{\alpha_{ii}} \qquad i = n, n-1, n-2, \ldots, 1$$

Convention: If lower index is greater than upper index do not execute sum

Gauss Elimination Algorithm I

- Store results of subtraction in existing a_{ij} variables
- Subtract a_{k1}/a_{11} times row 1 from row k = 2, n

$$a_{k,j} \leftarrow a_{k,j} - \frac{a_{k,1}}{a_{1,1}} a_{1,j} \qquad b_k \leftarrow b_k - \frac{a_{k,1}}{a_{1,1}} b_1 \qquad \underline{j = 2,n}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & \cdots & a_{1n} & b_1 \\ 0 & a_{22} & a_{23} & \cdots & \cdots & a_{2n} & b_2 \\ 0 & a_{32} & a_{33} & \cdots & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & a_{n2} & a_{n3} & \cdots & \cdots & a_{nn} & b_n \end{bmatrix}$$

Zeros not actually computed

k is row index, j is column index

Gauss Elimination Algorithm II

- Store results of subtraction in existing a_{ij} variables
- Subtract a_{k2}/a_{22} times row 2 from row k = 3, n

$$a_{k,j} \leftarrow a_{k,j} - \frac{a_{k,2}}{a_{2,2}} a_{2,j} \qquad b_k \leftarrow b_k - \frac{a_{k,2}}{a_{2,2}} b_2 \qquad \underline{j = 3,n}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & \cdots & a_{1n} & b_1 \\ 0 & a_{22} & a_{23} & \cdots & \cdots & a_{2n} & b_2 \\ 0 & 0 & a_{33} & \cdots & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & a_{n3} & \cdots & \cdots & a_{nn} & b_n \end{bmatrix}$$

Zeros not actually computed

k is row index, j is column index

Gauss Elimination Algorithm III

- Store results of subtraction in existing a_{ij} variables
- Subtract a_{k3}/a_{33} times row 3 from row k = 4, n

$$a_{k,j} \leftarrow a_{k,j} - \frac{a_{k,3}}{a_{3,3}} a_{3,j} \qquad b_k \leftarrow b_k - \frac{a_{k,3}}{a_{3,3}} b_3 \qquad \underline{j = 4,n}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & \cdots & a_{1n} & b_1 \\ 0 & a_{22} & a_{23} & \cdots & \cdots & a_{2n} & b_2 \\ 0 & 0 & a_{33} & \cdots & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & 0 & \ddots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & a_{nn} & b_n \end{bmatrix}$$

Zeros not actually computed

k is row index, j is column index

Gauss Elimination General III

- The general rules for any row as the pivot row are the following
- Use this formula:

$$a_{row,column} \leftarrow a_{row,column} - \frac{a_{row,pivot}}{a_{pivot,pivot}} a_{pivot,column}$$

- Outermost loop: use rows 1 to n – 1 as the pivot row
  - Next loop over all rows below the pivot row to subtract pivot row from rows below it
    - Innermost loop uses the general formula above to get a new value of $a_{row,column}$ for each column

Gauss Pseudocode

For pivot = 1 to n − 1

For row = pivot + 1 to n

For column = pivot + 1 to n

$$a_{row,column} \leftarrow a_{row,column} - \frac{a_{row,pivot}}{a_{pivot,pivot}} a_{pivot,column}$$

$$b_{row} \leftarrow b_{row} - \frac{a_{row,pivot}}{a_{pivot,pivot}} b_{pivot}$$

Handle b as part of augmented a matrix

Column limits above assume that there is no need to actually produce the zeros in lower part of the matrix

Gauss Pseudocode

For pivot = 1 to n − 1

For row = pivot + 1 to n

For column = pivot + 1 to n + 1

$$a_{row,column} \leftarrow a_{row,column} - \frac{a_{row,pivot}}{a_{pivot,pivot}} a_{pivot,column}$$

+1 in column loop handles b as part of augmented a matrix

Column limits above assume that there is no need to actually produce the zeros in lower part of the matrix

Gauss Pseudocode

$$For\ pivot = 1\ to\ n-1$$
$$\quad For\ row = pivot+1\ to\ n$$
$$\quad\quad For\ column = pivot+1\ to\ n+nRHS$$
$$\quad\quad\quad a_{row,column} \leftarrow a_{row,column} - \frac{a_{row,pivot}}{a_{pivot,pivot}} a_{pivot,column}$$

Can handle multiple right hand side vectors, b, in one solution. Here nRHS is number of b columns added to matrix. Need multiple back substitutions in this case.

Column limits above assume that there is no need to actually produce the zeros in lower part of the matrix

Multiple **b** Back Substitution

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}} \quad \Rightarrow \quad x_{row,bCol} = \frac{a_{row,bCol} - \sum_{col=row+1}^{n} a_{row,col} x_{col,bCol}}{a_{row,row}}$$

- Loop over all **b** columns, **bCol** = 1 to nRHS
  - Loop over all rows in reverse order: ***row*** = n, n – 1, n – 2, … , 2, 1
    - Set x(***row***, **bCol**) equal to a(***row***, n+**bCol**)
    - Loop over all columns, ***col*** = ***row***+1 to n
      - In column loop set x(***row***, **bCol**) = x(***row***, **bCol**) – a(***row***,***col***) * x(***col***,**bCol**)
    - At end of loop set x(***row***, **bCol**) = x(***row***, **bCol**) / a(***row***, ***row***)

Zero for $a_{pivot,pivot}$

- If we find a zero on the pivot row/column, we cannot compute $a_{row,pivot}/a_{pivot,pivot}$
- We can swap data from a row below the original pivot row with the original pivot row data
  - This is equivalent to changing the order of the equations, which does not change the solution of the problem
    - It does change he sign of the determinant
  - If we find that all the rows below the pivot row have zeros we have a singular matrix

What is a Singular Matrix?

- A singular matrix is one in which the rows are not linearly independent
  - Rank less than n for an n by n matrix
- Example below shows original matrix, **A**, and it's row echelon form
  - We see that rank(**A**) = 2 and we can show that row 3 of **A** = 2(row 1) – (row 2)/2

$$\boldsymbol{A} = \begin{bmatrix} 3 & 7 & -2 \\ -4 & 14 & 12 \\ 8 & 7 & -10 \end{bmatrix} \quad \tilde{\boldsymbol{A}} = \begin{bmatrix} 3 & 7 & -2 \\ 0 & 23.3 & 9.3 \\ 0 & 0 & 0 \end{bmatrix}$$

Singular Matrix Meaning

- If we find all zeros in the pivot column for the pivot row and below we do not have a unique solution
  - Recall that an nxn matrix had to have
    - Rank[**A**] = Rank[**A b**] = n for a unique solution
    - Rank[**A**] = Rank[**A b**] < n for infinite solutions
    - Rank[**A**] ≠ Rank[**A b**] for no solution
  - A singular matrix has Rank[**A**] < n so it may have no solution or infinite solutions
  - It will not have a unique solution

Accuracy Problems

- Real numbers on a computer are not represented exactly
  - Previously discussed binary representation
- Errors in initial representation of numbers will increase with number of calculations
  - This is known as error propagation
- Machine epsilon, ε, is smallest number for which $1 + \varepsilon \neq 1$
  - MATLAB command `eps` gives $\varepsilon = 2.204 \times 10^{-16}$
  - VBA Double has same value

Condition Number

- Matrix condition number defined as the product of two norms: $\|\mathbf{A}\|\ \|\mathbf{A}^{-1}\|$
  - Condition number close to 1 is well conditioned (less susceptible to rounding error)
  - MATLAB function cond(A) gives $\|\mathbf{A}\|\ \|\mathbf{A}^{-1}\|$
  - Any norm may be used, but the most common measure is the "2" norm
  - Recall MATLAB norm functions
    - norm(A,<kind>) where <kind> is 1, 2, inf, or 'fro' for one, two, infinity or Frobenius norm

Help for Improving Accuracy

- Example: Use only three significant figures to solve the 2x2 system below

System $0.01x_1 + 100x_2 = 100$, $x_1 + x_2 = 2$ Exact Solution $x_1 = \frac{10000}{9999}$, $x_2 = \frac{9998}{9999}$

- Solve by Gaussian elimination
  - Multiply first equation by 1 by 1/0.01 and subtract from second equation

$$\left[1 - \frac{1}{.01}.01\right]x_1 + \left[1 - \frac{1}{.01}100\right]x_2 = \left[2 - \frac{1}{.01}100\right]$$

$$(1 - 10000)x_2 = 2 - 10000$$

Help for Improving Accuracy II

- Previous result $(1-10000)x_2 = 2-10000$
- With 3 significant figures this becomes $-10000x_2 = -10000$ or $x_2 = 1$ which is close to correct answer *Error! Exact value ≈ 1*
- Back substitution in first equation gives

$$0.01x_1 + 100x_2 = 100 \quad \Rightarrow \quad x_1 = \frac{100 - 100x_2}{.01} = \frac{100 - 100(1)}{.01} = 0$$

- What happens if we change the order of the equations?

$$x_1 + x_2 = 2$$
$$0.01x_1 + 100x_2 = 100$$

Help for Improving Accuracy III

- Multiply first equation by .01/1 and subtract it from the second equation

$$x_1 + x_2 = 2$$
$$0.01x_1 + 100x_2 = 100$$

$$\left[.01 - \frac{.01}{1}1\right]x_1 + \left[100 - \frac{.01}{1}1\right]x_2 = \left[100 - \frac{.01}{1}2\right]$$

- Result is $(100 - .01)x_2 = 100 - .02$
- To three significant figures this is $x_2 = 1$
- Back substitution gives x_1 as follows

$$x_1 + x_2 = 2 \quad \Rightarrow \quad x_1 = \frac{2 - x_2}{1} = \frac{100 - (1)}{1} = 1$$

 x_1 and x_2 accurate to 3 significant figures 34

What Happened?

- When we changed the order of the equations, we improved the accuracy
- The change we made changed the a_{11} term from 0.01 to 1
- Recall the equation for each $a_{row,column}$

$$a_{row,column} \leftarrow a_{row,column} - \frac{a_{row,pivot}}{a_{pivot,pivot}} a_{pivot,column}$$

- A larger value of the pivot element, $a_{pivot,pivot}$, will reduce the value subtracted form the matrix data
  - This reduces the roundoff error

Pivoting Strategy

- Each time before we start with a new pivot row do the following
  - Search each row from the pivot row to the last row to find the maximum (in absolute value) element in the pivot column
  - Swap the row with this maximum element with the pivot row (if not already in pivot row)
    - This has no effect on the solution since we are only changing the order of the equations
    - Could also look for maximum element in columns, but this requires more complex logic with little extra benefit

Two Tasks in One

- When we search for the maximum element in the pivot column
  - We improve the accuracy of the solution by swapping the row with the maximum element with the original pivot row data
  - If we find that we do not have a nonzero element in the pivot column (to machine accuracy) in the pivot row and below we know that we do not have a unique solution
    - In this case we stop the solution process and exit with a "singular matrix" error message

Gaussian Elimination Pseudo Code

- Loop over all rows,1 to n–1, as pivot rows
  - Search for largest element in pivot column and swap rows if necessary
    - If you cannot find a nonzero value in the pivot column return error message "singular matrix"
  - Loop over all rows below the pivot row

Tests for x = 0 should test for abs(x) < ε

- Loop over all columns to the left of the pivot column (including the right-side b values)
  - Replace each column element, $a_{row,column}$, by the formula

$$a_{row,column} \leftarrow a_{row,column} - \frac{a_{row,pivot}}{a_{pivot,pivot}} a_{pivot,column}$$

Scaling

- Another technique for avoiding roundoff errors
  - After creating the [**A b**] matrix, but before doing any calculations
  - Loop over all rows of [**A b**]
    - For each row find the maximum element of **A** in absolute value
    - For that row, divide all $a_{row,column}$ values (including the b_{row} data) by this maximum value
  - This produces an array in which the maximum $a_{row,column}$ value in each row is one

Other Methods

- Gauss-Jordan turns original **A** matrix into a unit matrix
  - Resulting changes to the **b** column make it equal to the unknowns
- LU method creates a lower triangular and upper triangular matrix (**L** and **U**) such that **LU** = **A**
  - Once **LU** is formed new **b** columns can be solved for **x** by 2 back substitutions
- Modifications for special matrix structure

Methods Compared

- Gaussian before back substitution

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \cdots & \cdots & \alpha_{1n-1} & \alpha_{1n} \\ 0 & \alpha_{22} & \alpha_{23} & \cdots & \cdots & \alpha_{2n-1} & \alpha_{2n} \\ 0 & 0 & \alpha_{33} & \cdots & \cdots & \alpha_{3n-1} & \alpha_{3n} \\ \vdots & \vdots & \vdots & \ddots & & & \\ \vdots & \vdots & \vdots & & \ddots & & \\ 0 & 0 & 0 & \cdots & \cdots & \alpha_{n-1n-1} & \alpha_{n-1n} \\ 0 & 0 & 0 & \cdots & \cdots & 0 & \alpha_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix}$$

- Gauss-Jordan does not require back substitution

LU Method **A** = **LU**

Various forms – some use ones on **U** diagonal and l_{ii} on **L** diagonal

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & \cdots & a_{nn} \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ l_{n1} & l_{n2} & a_{n3} & \cdots & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & u_{nn} \end{bmatrix} = \mathbf{LU}$$

LU Components

- Get from matrix multiplication equations for **A** = **LU**, accounting for missing triangular parts in each matrix

Pattern: compute one row of **U** followed by one column of **L**

$$u_{1j} = a_{1j} \quad j = 1,n \qquad l_{k1} = \frac{a_{k1}}{u_{11}} \quad k = 2,n$$

$$u_{mj} = a_{mj} - \sum_{k=1}^{m-1} l_{mk} u_{kj} \qquad j = m,\ldots,n$$

$$l_{im} = \frac{a_{im} - \sum_{k=1}^{m-1} l_{ik} u_{km}}{u_{mm}} \qquad i = m+1,\ldots,n$$

LU Component Storage

Store **U** and **L** components in original **A** matrix; possible because we to not need to store ones on **L** diagonal

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & \cdots & u_{1n} \\ l_{21} & u_{22} & u_{23} & \cdots & \cdots & u_{2n} \\ l_{31} & l_{32} & u_{33} & \cdots & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & \cdots & l_{nn} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ l_{n1} & l_{n2} & a_{n3} & \cdots & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & u_{nn} \end{bmatrix} = \mathbf{LU}$$

LU Forward/Back Substitution

- From **Ax** = **b** and **A** = **LU**, **LUx** = **b**
- Define **y** = **Ux** so that **Ly** = **b**
- Use forward substitution to get y_i

$$y_i = b_i - \sum_{k=1}^{i-1} l_{ik} y_k \qquad i = 1,2,\ldots n \ (no\ sum\ \ for\ i = 1)$$

- Usual back substitution for x_i

$$x_i = \frac{b_i - \sum_{k=i+1}^{n} u_{ik} y_k}{a_{ii}} \qquad i = n, n-1, n-2,\ldots,2,1 \ (no\ sum\ \ for\ i = n)$$

Comparison of Methods

- Gaussian – basic approach, useful for one or a few **b** columns, smaller operation count
- Gauss-Jordan – higher operation count, simpler to perform, used to get inverse matrix by having n **b** columns that form a unit matrix
- LU – operation count similar to Gaussian – can get triangular matrices without knowing right-hand-sides

MATLAB Solvers

- To solve **Ax** = **b**, where **b** (and **x**) may have more than one column use x = A\b

```
>>A
A =  1   2   3
     4   5   6
     8   4   1
>>b
b = 14  16
    32  43
    19  45
```

```
>> x = A\b
x = 1   3
    2   5
    3   1
>> A*x - b
ans = 0   0
      0   0
      0   0
```

*x = A\b is more accurate than x = inv(A)*b*

MATLAB Solvers II

- `linsolve` function has several options to take advantage of special matrix structures
  - Can handle non-square matrices
  - Simplest form is linsolve(A,B)
  - Results below for previous A, b data

```
>> linsolve(A,b)
ans = 1     3
      2     5
      3     1
```

MATLAB Solvers III

- `lu` function creates L and U matrices that can be used to solve for x as new b data become available

```
>> [L U] = lu(A)
L = 0.125  0.500  1.000
    0.500  1.000      0
    1.000      0      0
U = 8.000  4.000  1.000
        0  3.000  5.500
        0      0  0.125
```


MATLAB Solvers IV

- Use L and U matrices for solution as shown previously

```
>> y=L\b
y =19.000  45.000
   22.500  20.500
    0.375   0.125
>> x=U\y
x =  1     3
     2     5
     3     1
```

Solution review:
For **LUx** = **b**
Define **y = Ux** so that **Ly** = **b**
Solve for **y** = $\mathbf{L}^{-1}\mathbf{b}$
Solve for **x** = $\mathbf{U}^{-1}\mathbf{y}$
Solutions with $\mathbf{L}^{-1}$ and $\mathbf{U}^{-1}$ "easy" because of triangular structures

What Kind of Solutions?

- Look at infinite solutions case

```
A =1   2  -3
   6 -12   8
  -1  10 -10
b = -4
     6
   -11
>> rank(A)
ans = 2
>> rank([A b])
ans = 2
>> x=A\b
Warning:
```

```
Matrix is singular to
working precision.
x =NaN
   NaN
   NaN
>> x=pinv(A)*b
x =-0.7433
   -0.2663
    0.9080
>> z = A*x-b
z =1e-014 * -0.1776
            -0.1776
            -0.3553
```

pinv is pseudo-inverse function; x is one of an infinity of solutions that has the minimum $||\mathbf{x}||_2$

The `rref` Function

- The `rref` function produces a reduced row-echelon form based on using the Gauss-Jordan method for the reduction

```
>> z=rref([A b])
z=1.0000       0  -0.8333   -1.5000
       0  1.0000  -1.0833   -1.2500
       0       0        0         0
```

- The expected result that Rank(A) = Rank([A b]) = 2 is found here
- This shows that $x_1 = 0.8333x_3 - 1.5$ and $x_2 = 1.0833x_3 - 1.25$ are solutions for any x_3
- Is previous solution consistent with this?

Overdetermined Systems

- Fit experimental data to $a_1x_i + a_2 = y_i$ $\mathbf{Aa} = \mathbf{y}$

$$\mathbf{x} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 20 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 0.5 \\ 2.2 \\ 5.6 \\ 9.8 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 5 & 1 \\ 10 & 1 \\ 20 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 2.2 \\ 5.6 \\ 9.8 \end{bmatrix}$$

```
A =  1     1
     5     1
    10     1
    20     1
>> rank(A)
ans = 2
>> rank([A y])
```

```
ans = 3
>> a=A\y
a = 0.4980
    0.0428
>> z = A*a
z = 0.5408
    2.5329
    5.0230
```

```
   5.0230
  10.0032
>> e = A*a-y
e = 0.0408
    0.3329
   -0.5770
    0.2032
```

repeated

Programming Assignment 4

- Look at MATLAB and Excel solvers using Gaussian elimination
- Use set of 100 equations whose exact solution is known
- Compare errors in solution by various methods
  - Modify existing VBA code to see effects of different data types and use or nonuse of pivoting

Programming Assignment 4 II

- MATLAB Operations
- Import data from Excel – note ability to import named ranges from Excel
  - MATLAB gives imported data the variable name data; be sure to give this another name (*e.g.* A = data) before importing more
  - Solve problem and compute RMS error between ***each*** MATLAB and known solution

$$E_{RMS} = \sqrt{\frac{\sum_{k=1}^{N}\left(x_{MATLAB,k} - x_{EXACT,k}\right)^2}{N}}$$

Programming Assignment 4 III

- Other MATLAB operations
  - Compute Det(**A**) and error in $\mathbf{AA}^{-1}$
  - Use MATLAB condition number function, `cond`, and norm function `normn`
  - Enter 4x4 **A** matrix and 4x1 **b** matrix that have infinite solutions and compare solutions using `pinv` and `rref`
- Solve system using Excel
  - Excel's mmult and minvese functions
  - VBA code version on next slide

Programming Assignment 4 IV

- Determine RMS errors in Excel
  - Using minverse
  - Using VBA code provided using double data type and pivoting
  - Modify previous code and use
    - Type double and no pivoting
    - Type single and no pivoting
    - Type single and pivoting
- Use Excel function mdeterm for Det(**A**)

$$E_{RMS} = \sqrt{\frac{\sum_{k=1}^{N}\left(x_{Excel,k} - x_{Exact,k}\right)^2}{N}}$$

Programming Assignment 4 V

- Modify Gaussian code to compute determinant instead of **x** solution
  - If **A** is converted to upper triangular form, Det(**A**) is product of diagonal elements a_{kk}
  - The sign of a determinant changes if rows are interchanged
    - Must modify code to keep track of each time a row is swapped
- See written assignment for discussion questions and files to submit

Quiz Four Solutions

- **A matrix, B, is a 3 by 5 matrix. How would you write the last element in the last row of B?**
- Since **B** has 3 rows and five columns the last element in the last row is b_{35}.
- **Find three components of the product FE: (a) first row first column, (b) last row, first column, and (c) last row, last column**
- $E = \begin{bmatrix} 3 & 5 & -6 & 10 \\ 12 & 7 & -9 & 4 \end{bmatrix}$ $F = \begin{bmatrix} 4 & 7 \\ -2 & 9 \\ 6 & 0 \end{bmatrix}$

Quiz Four Solutions II

- $E = \begin{bmatrix} 3 & 5 & -6 & 10 \\ 12 & 7 & -9 & 4 \end{bmatrix}$ $F = \begin{bmatrix} 4 & 7 \\ -2 & 9 \\ 6 & 0 \end{bmatrix}$
- The product matrix will have 3 rows (like **F**) and four columns (like **E**). The last element in the first row of the product, p_{14} = (4) (10) + (7)(4) = 68. The first element in the last row, p_{31} = (6)(3) + (0)(12) = 18; the last element in the last column, p_{34} = (6)(10) + (0)(4) = 60.

Quiz Four Solutions III

- For the **E** and F matrices in problem 2, which of the following matrix products are possible? **EF, E^TF, FE^T, EF^T E^TF^T, F^TE^T.**
- **E** has 4 columns and **F** has 3 rows so **EF** is **not** possible.
- $\mathbf{E}^T$ has 2 columns and **F** has 3 rows so $\mathbf{E}^T\mathbf{F}$ is **not** possible.
- $\mathbf{F}^T$ has 3 columns and **E** has 2 rows so $\mathbf{F}^T\mathbf{E}$ is **not** possible.
- **E** has 4 columns and $\mathbf{F}^T$ has 2 rows so $\mathbf{EF}^T$ is **not** possible.
- $\mathbf{E}^T$ has 2 columns and $\mathbf{F}^T$ has 2 rows so $\mathbf{E}^T\mathbf{F}^T$ **is possible**.
- $\mathbf{F}^T$ has 3 columns and $\mathbf{E}^T$ has 4 rows so $\mathbf{F}^T\mathbf{E}^T$ is **not** possible.

California State University Northridge 61

Quiz Four Solutions IV

- **Verify that the equation for A^{-1} is correct**

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad \mathbf{A}^{-1}=\frac{1}{a_{11}a_{22}-a_{21}a_{12}}\begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}$$

Do the matrix product ignoring the factor, $1/(a_{11}a_{22}+a_{21}a_{12})$, outside the matrix. First product, $p_{11} = a_{11}a_{22}+a_{12}(-a_{21})$ which is one when multiplied by the factor. Second product in the first row is $p_{11} = a_{11}(-a_{12})+a_{12}a_{11}=0$. First product in the second row is $p_{21} = a_{21}a_{22}+a_{22}(-a_{21})=0$. Final product in the second row is $p_{22} = a_{21}(-a_{12})+a_{22}(a_{11})$. When this is multiplied by the factor in front of the inverse, it becomes one.

California State University Northridge 62

Quiz Four solutions V

- **Apply Gauss elimination to the system of equations shown below. Find a unique solution if it exists; if not determine if there is no solution, or an infinite solution. If there is an infinite solution, find equations for x and y in terms of z.**

2x + 3y + z = 13
3x + 7y – 5z = 18
-x – 9y + 19z = –2

Multiply first equation by -1 and make it first equation; subtract 2 times this equation from the second equation and 3 times this equation from the third

California State University Northridge 63

Quiz Four Solutions VI

x + 9y – 19z = 2
-15y + 39z = 9
-20y + 52z = 12

Subtract (-20/-15) times the second equation from the third

x + 9y – 19z = 2
-15y + 39z = 9
0 + 0 = 0

Have an infinite solution. Second and third equations are equivalent. Use third equation to get y = (52z – 12) / 20
Substitute this into the first equation and manipulate to get x = (74 – 44z)/10.

California State University Northridge 64

Program Two Results

- Number of Students: 21
- Maximum possible score: 40
- Mean: 30.7
- Median: 31
- Standard deviation: 7.40
- Grade distribution:

10 15 26 26 27 27 30
30 31 31 31 32 32 34
34 36 36 38 39 39 40

California State University Northridge 65

Program Two Comments

- Spell MATLAB in all caps
- Use E notation for powers of 10
  - Prefer 3.4e8 to 3.4*10^8
    - Definitely do not use 3.4E10^8 = $(3.4\text{x}10^{10})^8$
- Do not copy intermediate output to Word
  - But copy your final results
- A script is different from a function
  - Script is like an Excel macro
- Look for unreasonable values or error flags like NaN
- Read directions carefully/ask questions

California State University Northridge 66