ECEN 615
Methods of Electric Power Systems Analysis

Lecture 6: Gaussian Elimination

Prof. Tom Overbye
Dept. of Electrical and Computer Engineering
Texas A&M University
overbye@tamu.edu

TEXAS A&M UNIVERSITY

Announcements

- If desired signup for the EPG dinner on Saturday September 20 at 5pm at Prof. Begovic's house
- Read Chapter 6
- Problem Set 1 is today.

Fast Decoupled Power Flow, cont.

- By continuing with Jacobian approximations we can obtain a reasonable approximation that is independent of the voltage magnitudes/angles.
  - This means the Jacobian need only be built/inverted once per power flow solution
- This approach is known as the fast decoupled power flow (FDPF)
- FDPF uses the same mismatch equations as standard power flow (just scaled) so it should have same solution
- The FDPF is widely used, though usually only for an approximate solution
  - Key fast decoupled power flow reference is B. Stott, O. Alsac, "Fast Decoupled Load Flow," *IEEE Trans. Power App. and Syst.*, May 1974, pp. 859-869
  - Modified versions also exist, such as D. Jajicic and A. Bose, "A Modification to the Fast Decoupled Power Flow for Networks with High R/X Ratios, "*IEEE Transactions on Power Sys.*, May 1988, pp. 743-746

FDPF Approximations

The FDPF makes the following approximations:

1. $|G_{ij}| = 0$
2. $|V_i| = 1$
3. $\sin\theta_{ij} = 0 \qquad \cos\theta_{ij} = 1$

To see the impact on the real power equations recall

$$P_i = \sum_{k=1}^{n} V_i V_k (G_{ik}\cos\theta_{ik} + B_{ik}\sin\theta_{ik}) = P_{Gi} - P_{Di}$$

Which can also be written as

$$\frac{P_i}{V_i} = \sum_{k=1}^{n} V_k (G_{ik}\cos\theta_{ik} + B_{ik}\sin\theta_{ik}) = \frac{P_{Gi} - P_{Di}}{V_i}$$

FDPF Approximations

- With the approximations for the diagonal term we get

$$\frac{\partial \mathrm{P_i}}{\partial \theta_i} \approx \sum_{\substack{k=1 \\ k \neq i}}^{n} B_{ik} = -B_{ii}$$

The for the off-diagonal terms (k ≠ i) with **G=0** and **V=1**

$$\frac{\partial P_i}{\partial \theta_k} = -B_{ik}\cos\theta_{ik} \approx -B_{ik}$$

- Hence the Jacobian for the real equations can be approximated as –**B**

FPDF Approximations

- For the reactive power equations we also scale by V_i

$$Q_i = \sum_{k=1}^{n} |V_i||V_k|(G_{ik}\sin\theta_{ik} - B_{ik}\cos\theta_{ik}) = Q_{Gi} - Q_{Di}$$

$$\frac{Q_i}{V_i} = \sum_{k=1}^{n} V_k(G_{ik}\sin\theta_{ik} - B_{ik}\cos\theta_{ik}) = \frac{Q_{Gi} - Q_{Di}}{V_i}$$

- For the Jacobian off-diagonals we get

$$\frac{\partial Q_i}{\partial V_k} = -B_{ik}\cos\theta_{ik} \approx -B_{ik}$$

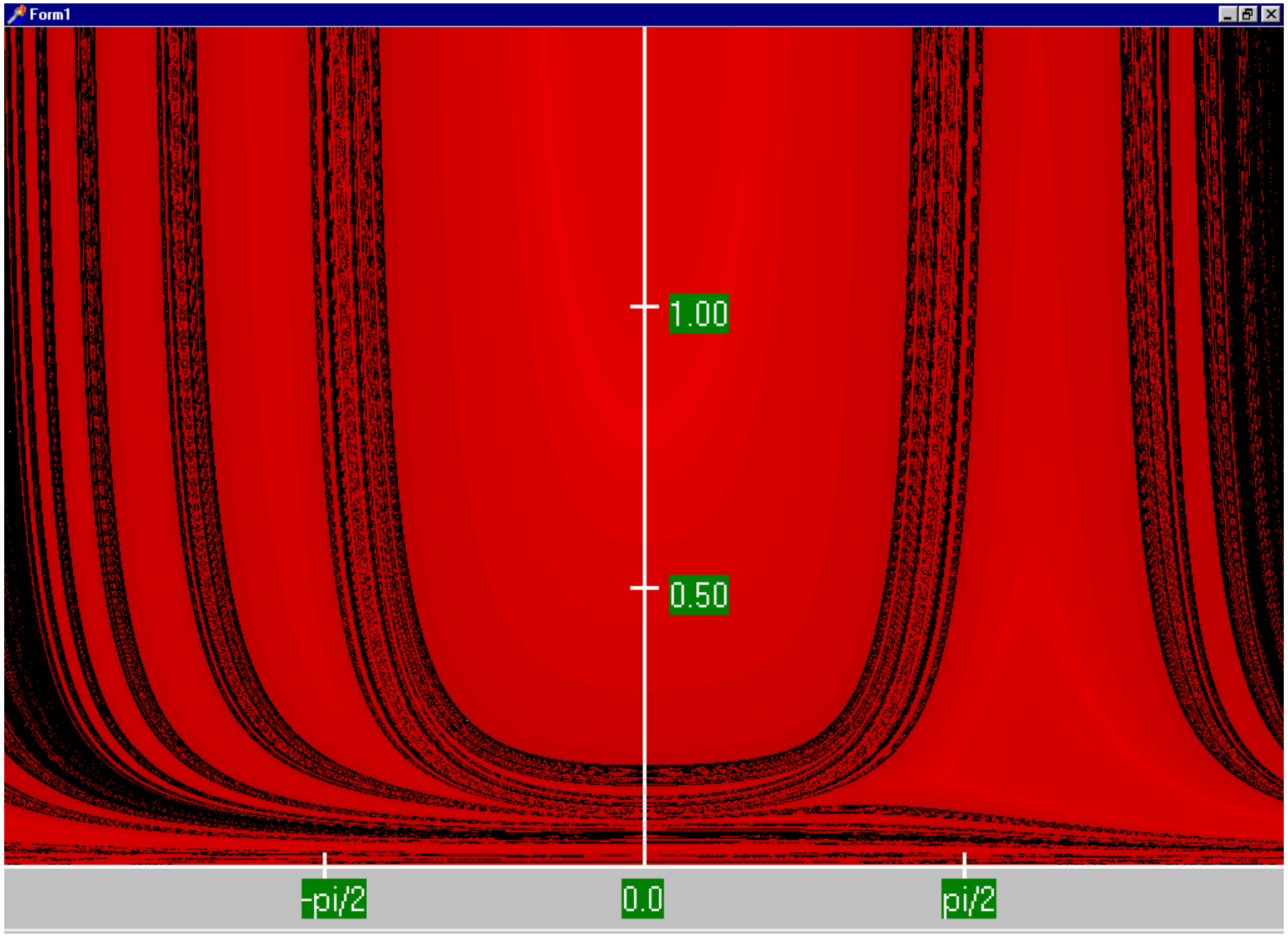
FDPF Approximations

- And for the reactive power Jacobian diagonal we get

$$\frac{\partial Q_i}{\partial V_i} \approx -2B_{ii} - \sum_{\substack{k=1 \\ k\neq i}}^{n} B_{ik} = -B_{ii}$$

- As derived the real and reactive equations have a constant Jacobian equal to –**B**
  - Usually modifications are made to omit from the real power matrix elements that affect reactive flow (like shunts) and from the reactive power matrix elements that affect real power flow, like phase shifters
  - We'll call the real power matrix **B**' and the reactive **B**"

FDPF Region of Convergence



FDPF Cautions

- The FDPF works well as long as the previous approximations hold for the entire system
- With the movement towards modeling larger systems, with more of the lower voltage portions of the system represented (for which r/x ratios are higher) it is quite common for the FDPF to get stuck because small portions of the system are ill-behaved
- The FDPF is commonly used to provide an initial guess of the solution for contingency analysis

DC Power Flow

- The "DC" power flow makes the most severe approximations:
  - completely ignore reactive power, assume all the voltages are always 1.0 per unit, ignore line conductance
- This makes the power flow a linear set of equations, which can be solved directly

$$\boldsymbol{\theta} = -\mathbf{B}^{-1}\ \mathbf{P}$$

P sign convention is generation is positive

- The term dc power flow actually dates from the time of the old network analyzers (going back into the 1930's)
- Not to be confused with the inclusion of HVDC lines in the standard NPF

DC Power Flow References

- I don't think a classic dc power flow paper exists; a nice formulation is given in our book *Power Generation and Control* book by Wood, Wollenberg and Sheble
- The August 2009 paper in IEEE Transactions on Power Systems, "DC Power Flow Revisited" (by Stott, Jardim and Alsac) provides good coverage
- T. J. Overbye, X. Cheng, and Y. Sun, "A comparison of the AC and DC power flow models for LMP Calculations," in *Proc. 37th Hawaii Int. Conf. System Sciences*, 2004, compares the accuracy of the approach

DC Power Flow Example

EXAMPLE 6.17

Determine the dc power flow solution for the five bus from Example 6.9.

SOLUTION With bus 1 as the system slack, the **B** matrix and **P** vector for this system are

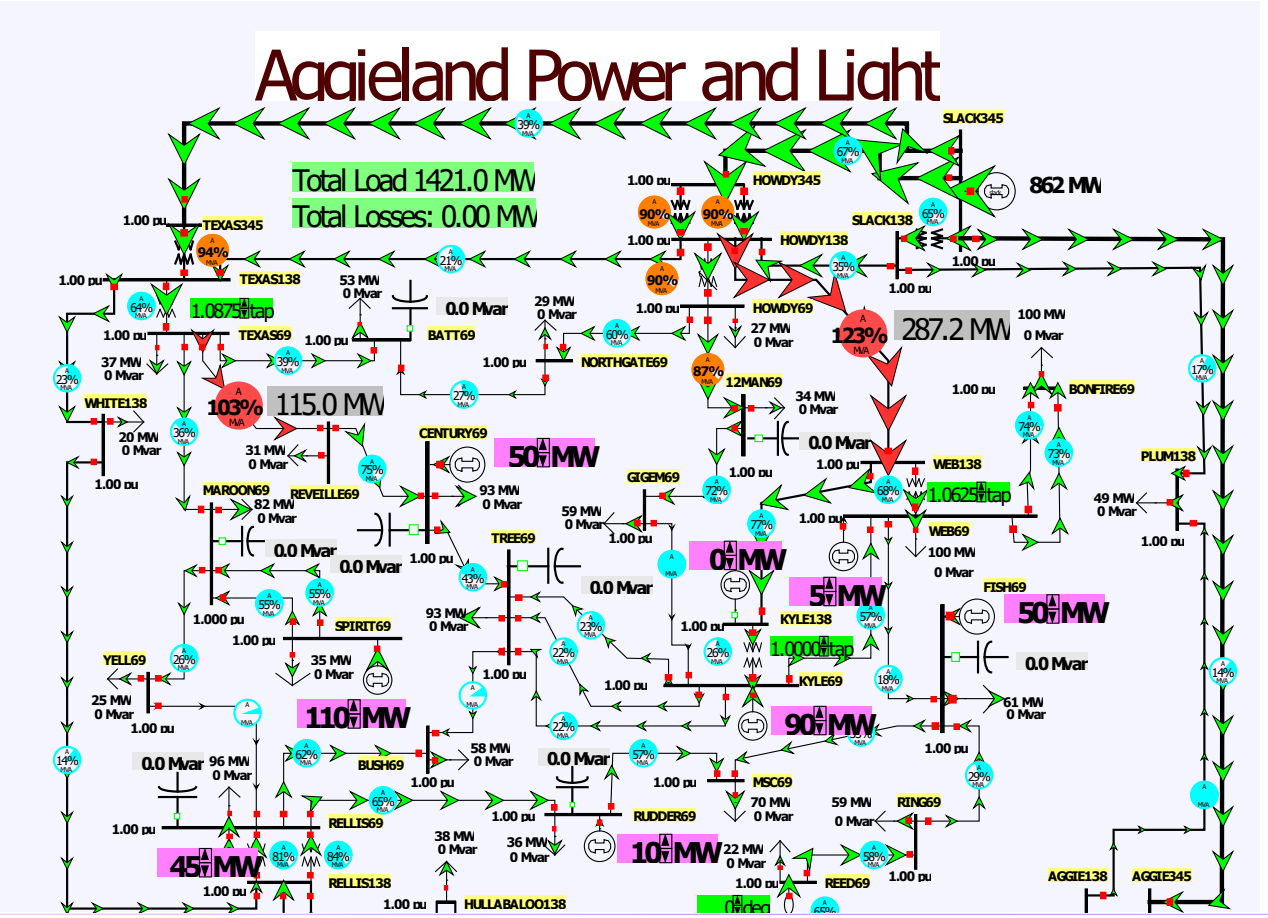
$$\mathbf{B} = \begin{bmatrix} -30 & 0 & 10 & 20 \\ 0 & -100 & 100 & 0 \\ 10 & 100 & -150 & 40 \\ 20 & 0 & 40 & -110 \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} -8.0 \\ 4.4 \\ 0 \\ 0 \end{bmatrix}$$

$$\boldsymbol{\delta} = -\mathbf{B}^{-1}\mathbf{P} = \begin{bmatrix} -0.3263 \\ 0.0091 \\ -0.0349 \\ -0.0720 \end{bmatrix} \text{ radians} = \begin{bmatrix} -18.70 \\ 0.5214 \\ -2.000 \\ -4.125 \end{bmatrix} \text{ degrees}$$

Example from Power System Analysis and Design, by Glover, Overbye, Sarma, 6th Edition

DC Power Flow in PowerWorld

- PowerWorld allows for easy switching between the dc and ac power flows (case **Aggieland37**)



To use the dc approach in PowerWorld select **Tools, Solve, DC Power Flow**

Notice there are no losses

Linear System Solution: Introduction

- A problem that occurs in many is fields is the solution of linear systems $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is an n by n matrix with elements a_{ij}, and $\mathbf{x}$ and $\mathbf{b}$ are n-vectors with elements x_i and b_i respectively
- In power systems we are particularly interested in systems when n is relatively large and $\mathbf{A}$ is sparse
  - How large is large is changing
- A matrix is sparse if a large percentage of its elements have zero values
- Goal is to understand the computational issues (including complexity) associated with the solution of these systems

Introduction, cont.

- Sparse matrices arise in many areas, and can have domain specific structures
  - Symmetric matrices
  - Structurally symmetric matrices
  - Tridiagnonal matrices
  - Banded matrices
- A good (and free) book on sparse matrices is available at www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf
- ECEN 615 is focused on problems in the electric power domain; it is not a general sparse matrix course
  - Much of the early sparse matrix work was done in power!

Gaussian Elimination

- The best known and most widely used method for solving linear systems of algebraic equations is attributed to Gauss
- Gaussian elimination avoids having to explicitly determine the inverse of **A**, which is $O(n^3)$
- Gaussian elimination can be readily applied to sparse matrices
- Gaussian elimination leverages the fact that scaling a linear equation does not change its solution, nor does adding on linear equation to another

$$2x_1 + 4x_2 = 10 \rightarrow x_1 + 2x_2 = 5$$

Gaussian Elimination, cont.

- Gaussian elimination is the elementary procedure in which we use the first equation to eliminate the first variable from the last n-1 equations, then we use the new second equation to eliminate the second variable from the last n-2 equations, and so on
- After performing n-1 such eliminations we end up with a triangular system which is easily solved in a backward direction

Example 1

- We need to solve for $\mathbf{x}$ in the system

$$\begin{bmatrix} 2 & 3 & -1 & 0 \\ -6 & -5 & 0 & 2 \\ 2 & -5 & 6 & -6 \\ 4 & 2 & 2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 20 \\ -45 \\ -3 \\ 30 \end{bmatrix}$$

- The three elimination steps are given on the next slides; for simplicity, we have appended the r.h.s. vector to the matrix
- First step is set the diagonal element of row 1 to 1 (i.e., normalize it)

Example 1, cont.

- Eliminate x_1 by subtracting row 1 from all the rows below it

multiply row 1 by $\frac{1}{2}$

multiply row 1 by 6 and add to row 2

multiply row 1 by -2 and add to row 3

multiply row 1 by -4 and add to row 4

$$\left[\begin{array}{cccc|c} 1 & \frac{3}{2} & -\frac{1}{2} & 0 & 10 \\ 0 & 4 & -3 & 2 & 15 \\ 0 & -8 & 7 & -6 & -23 \\ 0 & -4 & 7 & -3 & -10 \end{array}\right]$$

Example 1, cont.

- Eliminate x_2 by subtracting row 2 from all the rows below it

multiply row 2 by $\frac{1}{4}$

multiply row 2 by 8 and add to row 3

multiply row 2 by 4 and add to row 4

$$\left[\begin{array}{cccc|c} 1 & \frac{3}{2} & -\frac{1}{2} & 0 & 10 \\ 0 & 1 & -\frac{3}{4} & \frac{1}{2} & \frac{15}{4} \\ 0 & 0 & 1 & -2 & 7 \\ 0 & 0 & 1 & -1 & 5 \end{array}\right]$$

Example 1, cont.

- Elimination of x_3 from row 3 and 4

$$\left[\begin{array}{cccc|c} 1 & \frac{3}{2} & -\frac{1}{2} & 0 & 10 \\ 0 & 1 & -\frac{3}{4} & \frac{1}{2} & \frac{15}{4} \\ 0 & 0 & 1 & -2 & 7 \\ 0 & 0 & 0 & 1 & -2 \end{array}\right]$$

multiply row 3 by 1

multiply row 3 by -1 and add to row 4

Example 1, cont.

- Then, we solve for x by "going backwards", i.e., using back substitution:

$$x_4 = -2$$

$$x_3 - 2x_4 = 7 \Rightarrow x_3 = 3$$

$$x_2 - \frac{3}{4}x_3 + \frac{1}{2}x_4 = \frac{15}{4} \Rightarrow x_2 = 7$$

$$x_1 + \frac{3}{2}x_2 - \frac{1}{2}x_3 = 10 \Rightarrow x_1 = 1$$

LU Decomposition

- What we did with Gaussian elimination can be thought of as changing the form of the matrix to create two matrices with special structure
- One matrix, shown on the last slide, is upper triangular
- The second matrix, a lower triangular one, keeps track of the operations we did to get the upper triangular matrix
- These concepts will be helpful for a computer implementation of the algorithm and for its application to sparse systems

LU Decomposition Theorem

- Any nonsingular matrix **A** has the following factorization:

$$\mathbf{A} = \mathbf{LU}$$

where **U** could be the upper triangular matrix previously developed (with 1's on its diagonals) and **L** is a lower triangular matrix defined by

$$\ell_{ij} = \begin{cases} a_{ij}^{(j-1)} & j \le i \\ 0 & j > i \end{cases}$$

LU Decomposition Application

- As a result of this theorem we can rewrite

  $\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$

  Define $\mathbf{y} = \mathbf{Ux}$

  Then $\mathbf{Ly} = \mathbf{b}$

- Can also be set so **U** has non unity diagonals
- Once **A** has been factored, we can solve for **x** by first solving for **y**, a process known as forward substitution, then solving for **x** in a process known as back substitution
- In the previous example we can think of **L** as a record of the forward operations preformed on **b**.

LDU Decomposition

- In the previous case we required that the diagonals of **U** be unity, while there was no such restriction on the diagonals of **L**
- An alternative decomposition is

$$\mathbf{A} = \tilde{\mathbf{L}}\mathbf{D}\mathbf{U}$$

with $\mathbf{L} = \tilde{\mathbf{L}}\mathbf{D}$

where **D** is a diagonal matrix, and the lower triangular matrix is modified to require unity for the diagonals (we'll just use the **LU** approach in 615)

Symmetric Matrix Factorization

- The LDU formulation is quite useful for the case of a symmetric matrix

$$\mathbf{A} = \mathbf{A}^T$$

$$\mathbf{A} = \tilde{\mathbf{L}}\mathbf{D}\mathbf{U} = \mathbf{U}^T\mathbf{D}\tilde{\mathbf{L}}^T = \mathbf{A}^T$$

$$\mathbf{U} = \tilde{\mathbf{L}}^T$$

$$\mathbf{A} = \mathbf{U}^T\mathbf{D}\mathbf{U}$$

- Hence only the upper triangular elements and the diagonal elements need to be stored, reducing storage by almost a factor of 2

Symmetric Matrix Factorization

- There are also some computational benefits from factoring symmetric matrices. However, since symmetric matrices are not common in power applications, we will not consider them in-depth
- However, topologically symmetric sparse matrices are quite common, so those will be our main focus

Pivoting

- An immediate problem that can occur with Gaussian elimination is the issue of zeros on the diagonal; for example

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

- This problem can be solved by a process known as "pivoting," which involves the interchange of either both rows and columns (full pivoting) or just the rows (partial pivoting)
  - Partial pivoting is much easier to implement, and actually can be shown to work quite well

Pivoting, cont.

- In the previous example the (partial) pivot would just be to interchange the two rows

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

  obviously we need to keep track of the interchanged rows!
- Partial pivoting can be helpful in improving numerical stability even when the diagonals are not zero
  - When factoring row k interchange rows so the new diagonal is the largest element in column k for rows j >= k

LU Algorithm Without Pivoting Processing by row

- We will use the more common approach of having ones on the diagonals of **L**. Also in the common, diagonally dominant power system problems pivoting is not needed.
  The below algorithm is in row form (useful with sparsity!)

```
For i := 2 to n Do Begin  // This is the row being processed
  For j := 1 to i-1 Do Begin  // Rows subtracted from row i
    A[i,j] = A[i,j]/A[j,j]  // This is the scaling
    For k := j+1 to n Do Begin  // Go through each column in i
      A[i,k] = A[i,k] - A[i,j]*A[j,k]
    End;
  End;
End;
```

LU Example

- Starting matrix

$$\mathbf{A} = \begin{bmatrix} 20 & -12 & -5 \\ -5 & 12 & -6 \\ -4 & -3 & 8 \end{bmatrix}$$

- First row is unchanged; start with i=2
- Result with i=2, j=1; done with row 2

$$\mathbf{A} = \begin{bmatrix} 20 & -12 & -5 \\ -0.25 & 9 & -7.25 \\ -4 & -3 & 8 \end{bmatrix}$$

A[2,2]= A[2,2]-A[2,1]*A[1,2]
=12-(-0.25)*(-12) =9
A[2,3] = A[2,3]-A[2,1]*A[1,3]
=-6 –(-0.25)*(-5) = -7.25

LU Example, cont.

- Result with i=3, j=1;

$$\mathbf{A} = \begin{bmatrix} 20 & -12 & -5 \\ -0.25 & 9 & -7.25 \\ -0.2 & -5.4 & 7 \end{bmatrix}$$

A[3,1]= A[3,1]/A[1,1]

=-4/20= -0.2

A[3,2] = A[3,2] – A[3,1]*A[1,2]

A[3,2] = -3 – (-0.2)*(-12) = -5.4

A[3,3] = 8 – (-0.2)*(-5) = 7

- Result with i=3, j=2; done with row 3; done!

$$\mathbf{A} = \begin{bmatrix} 20 & -12 & -5 \\ -0.25 & 9 & -7.25 \\ -0.2 & -0.6 & 2.65 \end{bmatrix}$$

A[3,2]= A[3,2]/A[2,2]

=-5.4/9= -0.6

A[3,3] = A[3,3] – A[3,2]*A[2,3]

A[3,3] = 7 – (-0.6)*(-7.25) =2.65

LU Example, cont.

- Original matrix is used to hold **L** and **U**

$$\mathbf{A} = \begin{bmatrix} 20 & -12 & -5 \\ -5 & 12 & -6 \\ -4 & -3 & 8 \end{bmatrix} = \mathbf{LU}$$

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -0.25 & 1 & 0 \\ -0.2 & -0.6 & 1 \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} 20 & -12 & -5 \\ 0 & 9 & -7.25 \\ 0 & 0 & 2.65 \end{bmatrix}$$

With this approach the original **A** matrix has been replaced by the factored values!

Forward Substitution

Forward substitution solves **b = Ly** with values in **b** being over written (replaced by the **y** values)

```
For i := 2 to n Do Begin  // This is the row being processed
  For j := 1 to i-1 Do Begin
    b[i] = b[i] - A[i,j]*b[j]    // This is just using the L matrix
  End;
End;
```

Forward Substitution Example

Let $\mathbf{b} = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$

From before $\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -0.25 & 1 & 0 \\ -0.2 & -0.6 & 1 \end{bmatrix}$

$$y[1] = 10$$

$$y[2] = 20 - (-0.25) * 10 = 22.5$$

$$y[3] = 30 - (-0.2) * 10 - (-0.6) * 22.5 = 45.5$$

Backward Substitution

- Backward substitution solves **y** = **Ux** (with values of **y** contained in the **b** vector as a result of the forward substitution)

```
For i := n to 1 Do Begin  // This is the row being processed
  For j := i+1 to n Do Begin
    b[i] = b[i] - A[i,j]*b[j]    // This is just using the U matrix
  End;
  b[i] = b[i]/A[i,i]    // The A[i,i] values are <> 0 if it is nonsingular
End
```

Backward Substitution Example

Let $\mathbf{y} = \begin{bmatrix} 10 \\ 22.5 \\ 45.5 \end{bmatrix}$

From before $\mathbf{U} = \begin{bmatrix} 20 & -12 & -5 \\ 0 & 9 & -7.25 \\ 0 & 0 & 2.65 \end{bmatrix}$

$$x[3] = (1/2.65) * 45.5 = 17.17$$

$$x[2] = (1/9) * \left(22.5 - (-7.25) * 17.17\right) = 16.33$$

$$x[1] = (1/20) * \left(10 - (-5) * 17.17 - (-12) * 16.33\right) = 14.59$$