ECEN 615
Methods of Electric Power Systems Analysis

Lecture 7: DC Power Flow, Gaussian Elimination, Sparse Systems

---

Prof. Tom Overbye

Dept. of Electrical and Computer Engineering

Texas A&M University

overbye@tamu.edu

Announcements

- Read Chapter 6 from the book
  - The book does the power flow using the polar form for the Y_{bus} elements
- Homework 2 is due on Thursday September 17
- For homework 2 you'll need to commercial version of PowerWorld Simulator.

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

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

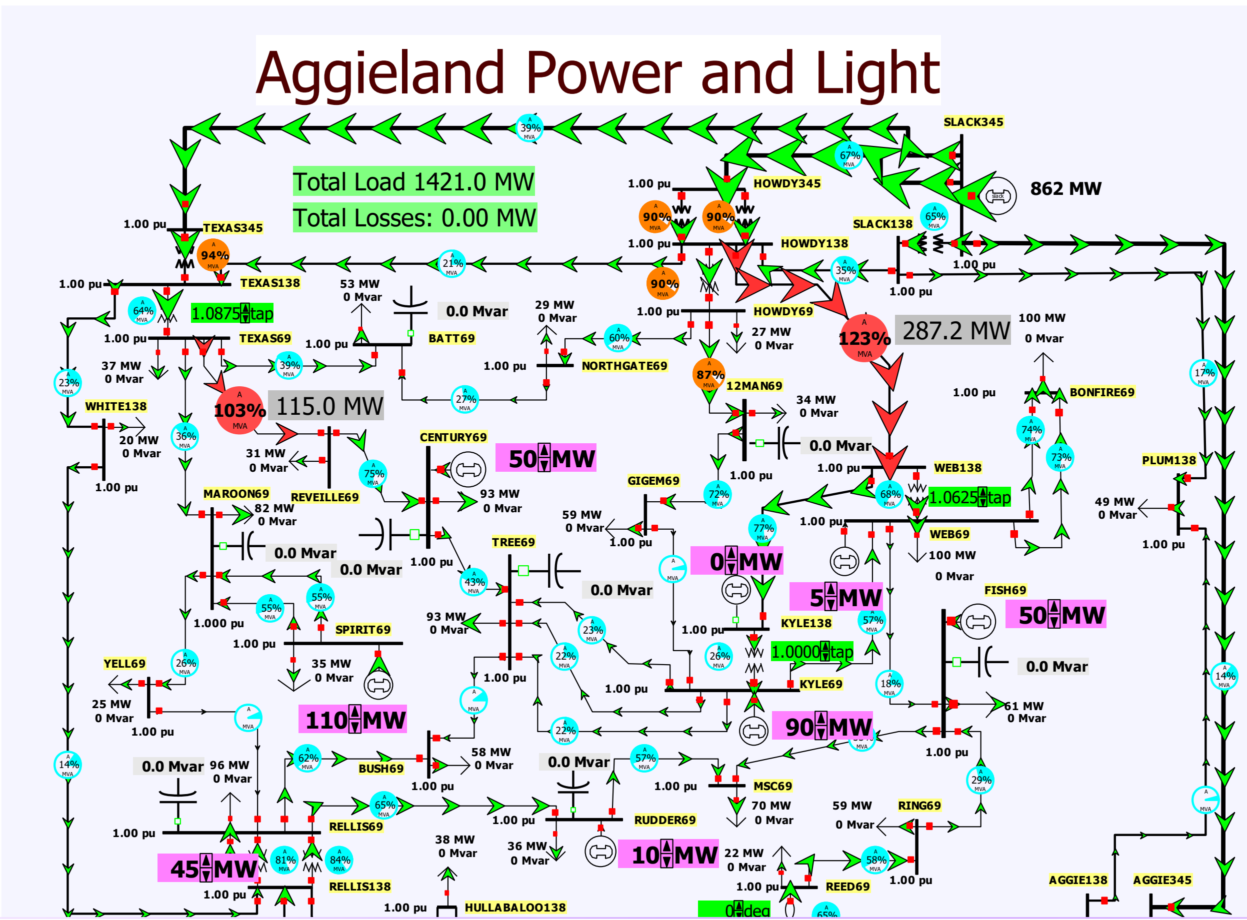
$$\boldsymbol{\delta} = -\mathbf{B}^{-1}\mathbf{P} = \begin{bmatrix} -0.3263 \\ 0.0091 \\ -0.0349 \\ -0.0720 \end{bmatrix} \text{radians} = \begin{bmatrix} -18.70 \\ 0.5214 \\ -2.000 \\ -4.125 \end{bmatrix} \text{degrees}$$

The output of the generator at bus 3 is now 440 MW

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 **Ax = b** where **A** is an n by n matrix with elements a_{ij}, and **x** and **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 **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 **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

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

Example 1, cont.

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

$$\text{multiply row 2 by } \frac{1}{4} \qquad \text{multiply row 2 by 8 and add to row 3} \qquad \text{multiply row 2 by 4 and add to row 4}$$

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

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

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

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

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

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

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

$$y[1] = 10$$

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

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

Backward Substitution

- Backward substitution solves $\mathbf{y} = \mathbf{U}\mathbf{x}$ (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

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

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

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

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

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

Computational Complexity

- Computational complexity indicates how the number of numerical operations scales with the size of the problem
- Computational complexity is expressed using the "Big O" notation; assume a problem of size n
  - Adding the number of elements in a vector is O(n)
  - Adding two n by n full matrices is $O(n^2)$
  - Multiplying two n by n full matrices is $O(n^3)$
  - Inverting an n by n full matrix, or doing Gaussian elimination is $O(n^3)$
  - Solving the traveling salesman problem by brute-force search is O(n!)

Computational Complexity

- Knowing the computational complexity of a problem can help to determine whether it can be solved (at least using a particular method)
  - Scaling factors do not affect the computation complexity
    - an algorithm that takes $n^3/2$ operations has the same computational complexity of one the takes $n^3/10$ operations (though obviously the second one is faster!)
- With $O(n^3)$ factoring a full matrix becomes computationally intractable quickly!
  - A 100 by 100 matrix takes a million operations (give or take)
  - A 1000 by 1000 matrix takes a billion operations
  - A 10,000 by 10,000 matrix takes a trillion operations!

Sparse Systems

- The material presented so far applies to any arbitrary linear system
- The next step is to see what happens when we apply triangular factorization to a sparse matrix
- For a sparse system, only nonzero elements need to be stored in the computer since no arithmetic operations are performed on the 0's
- The LU factorization is adapted to solve sparse systems in such a way as to preserve the sparsity as much as possible

Sparse Matrix History

- A nice overview of sparse matrix history is by Iain Duff at http://www.siam.org/meetings/la09/talks/duff.pdf
- Sparse matrices developed simultaneously in several different disciplines in the early 1960’s with power systems definitely one of the key players (Bill Tinney from BPA)
- Different disciplines claim credit since they didn’t necessarily know what was going on in the others

Sparse Matrix History

- In power systems a key N. Sato, W.F. Tinney, “Techniques for Exploiting the Sparsity of the Network Admittance Matrix,” Power App. and Syst., pp 944-950, December 1963
  - In the paper they are proposing solving systems with up to 1000 buses (nodes) in 32K of memory!
  - You’ll also note that in the discussion by El-Abiad, Watson, and Stagg they mention the creation of standard test systems with between 30 and 229 buses (this surely included the now famous 118 bus system)
  - The BPA authors talk “power flow” and the discussors talk “load flow.”
- Tinney and Walker present a much more detailed approach in their 1967 IEEE Proceedings paper titled “Direct Solutions of Sparse Network Equations by Optimally Order Triangular Factorization”

Sparse Matrix Computational Order

- The computational order of factoring a sparse matrix, or doing a forward/backward substitution depends on the matrix structure
  - Full matrix is $O(n^3)$
  - A diagonal matrix is $O(n)$; that is, just invert each element
- For power system problems the classic paper is F. L. Alvarado, "Computational complexity in power systems," *IEEE Transactions on Power Apparatus and Systems*, ,May/June 1976
  - $O(n^{1.4})$ for factoring, $O(n^{1.2})$ for forward/backward
  - For a 100,000 by 100,000 matrix changes computation for factoring from 1 quadrillion to 10 million!

Inverse of a Sparse Matrix

- The inverse of a sparse matrix is NOT in general a sparse matrix
- We never (or at least very, very, very seldom) explicitly invert a sparse matrix
  - Individual columns of the inverse of a sparse matrix can be obtained by solving $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ with $\mathbf{b}$ set to all zeros except for a single nonzero in the position of the desired column
  - If a few desired elements of $\mathbf{A}^{-1}$ are desired (such as the diagonal values) they can usually be computed quite efficiently using sparse vector methods (a topic we'll be considering soon)
- We can't invert a singular matrix (whether sparse or full)

Computer Architecture Impacts

- With modern computers the processor speed is many times faster than the time it takes to access data in main memory
  - Some instructions can be processed in parallel
- Caches are used to provide quicker access to more commonly used data
  - Caches are smaller than main memory
  - Different cache levels are used with the quicker caches, like L1, have faster speeds but smaller sizes; L1 might be 256K, whereas the slower L2 might be 2M
- Data structures can have a significant impact on sparse matrix computation

Full Matrix versus Sparse Matrix Storage

- Full matrices are easily stored in arrays with just one variable needed to store each value since the value's row and column are implicitly available from its matrix position
- With sparse matrices two or three elements are needed to store each value
  - The zero values are not explicitly stored
  - The value itself, its row number and its column number
  - Storage can be reduced by storing all the elements in a particular row or column together
- Because large matrices are often quite sparse, the total storage is still substantially reduced

Sparse Matrix Usage Can Determine the Optimal Storage

- How a sparse matrix is used can determine the best storage scheme to use
  - Row versus column access; does the structure change
- Is the matrix essentially used only once? That is, its structure and values are assumed new each time used
- Is the matrix structure constant, with its values changed
  - This would be common in the N-R power flow, in which the structure doesn't change each iteration, but its values do
- Is the matrix structure and values constant, with just the **b** vector in **Ax=b** changing
  - Quite common in transient stability solutions

Numerical Precision

- Required numerical precision determines type of variables used to represent numbers
  - Specified as number of bytes, and whether signed or not
- For Integers
  - One byte is either 0 to 255 or -128 to 127
  - Two bytes is either smallint (-32,768 to 32,767) or word (0 to 65,536)
  - Four bytes is either Integer (-2,147,483,648 to 2,147,483,647) or Cardinal (0 to 4,294,967,295)
    - This is usually sufficient for power system row/column numbers
  - Eight bytes (Int64) if four bytes is not enough

Numerical Precision, cont.

- For floating point values using choice is between four bytes (single precision) or eight bytes (double precision); extended precision has ten bytes
  - Single precision allows for 6 to 7 significant digits
  - Double precision allows for 15 to 17 significant digits
  - Extended allows for about 18 significant digits
  - More bytes requires more storage
  - Computational impacts depend on the underlying device; on PCs there isn’t much impact; GPUs can be 3 to 8 times slower for double precision
- For most power problems double precision is best

General Sparse Matrix Storage

- A general approach for storing a sparse matrix would be using three vectors, each dimensioned to number of elements
  - AA: Stores the values, usually in power system analysis as double precision values (8 bytes)
  - JR: Stores the row number; for power problems usually as an integer (4 bytes)
  - JC: Stores the column number, again as an integer
- If unsorted then both row and column are needed
- New elements could easily be added, but costly to delete
- Unordered approach doesn't make for good computation since elements used next computationally aren't necessarily nearby
- Usually ordered, either by row or column

Sparse Storage Example

- Assume

$$\mathbf{A} = \begin{bmatrix} 5 & 0 & 0 & -4 \\ 0 & 4 & 0 & -3 \\ 0 & 0 & 3 & -2 \\ -4 & -3 & -2 & 10 \end{bmatrix}$$

- Then

$$\mathbf{AA} = \begin{bmatrix} 5 & -4 & 4 & -3 & 3 & -2 & -4 & -3 & -2 & 10 \end{bmatrix}$$

$$\mathbf{JR} = \begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 3 & 4 & 4 & 4 & 4 \end{bmatrix}$$

$$\mathbf{JC} = \begin{bmatrix} 1 & 4 & 2 & 4 & 3 & 4 & 1 & 2 & 3 & 4 \end{bmatrix}$$

Note, this example is a symmetric matrix, but the technique is general