

# EE 521/ECE 582 – Analysis of Power systems

Class #6 – September 13, 2022

*Dr. Noel N. Schulz*  
*Edmund O. Schweitzer III Chair in*  
*Power Apparatus and Systems*  
*Chief Scientist Joint Appointment, PNNL*  
*Co-Director, PNNL/WSU Advanced Grid Institute (AGI)*  
*Washington State University Pullman*  
*[Noel.Schulz@wsu.edu](mailto:Noel.Schulz@wsu.edu) EME 35*  
*509-335-0980 (o) and 509-336-5522 (c)*

## Student Hours

- Pullman this week
  - Tuesdays 4:30-5:30 pm (after class)
  - Wednesdays 4-5 pm
  - Fridays 1:30-2:30 pm
- Join from PC, Mac, Linux, iOS, or Android: <https://wsu.zoom.us/j/8237216735> (Links to an external site.)
- Meeting ID: 823 721 6735

Or by request via email [noel.Schulz@wsu.edu](mailto:noel.Schulz@wsu.edu) or phone 509-336-5522

## Matlab Resources

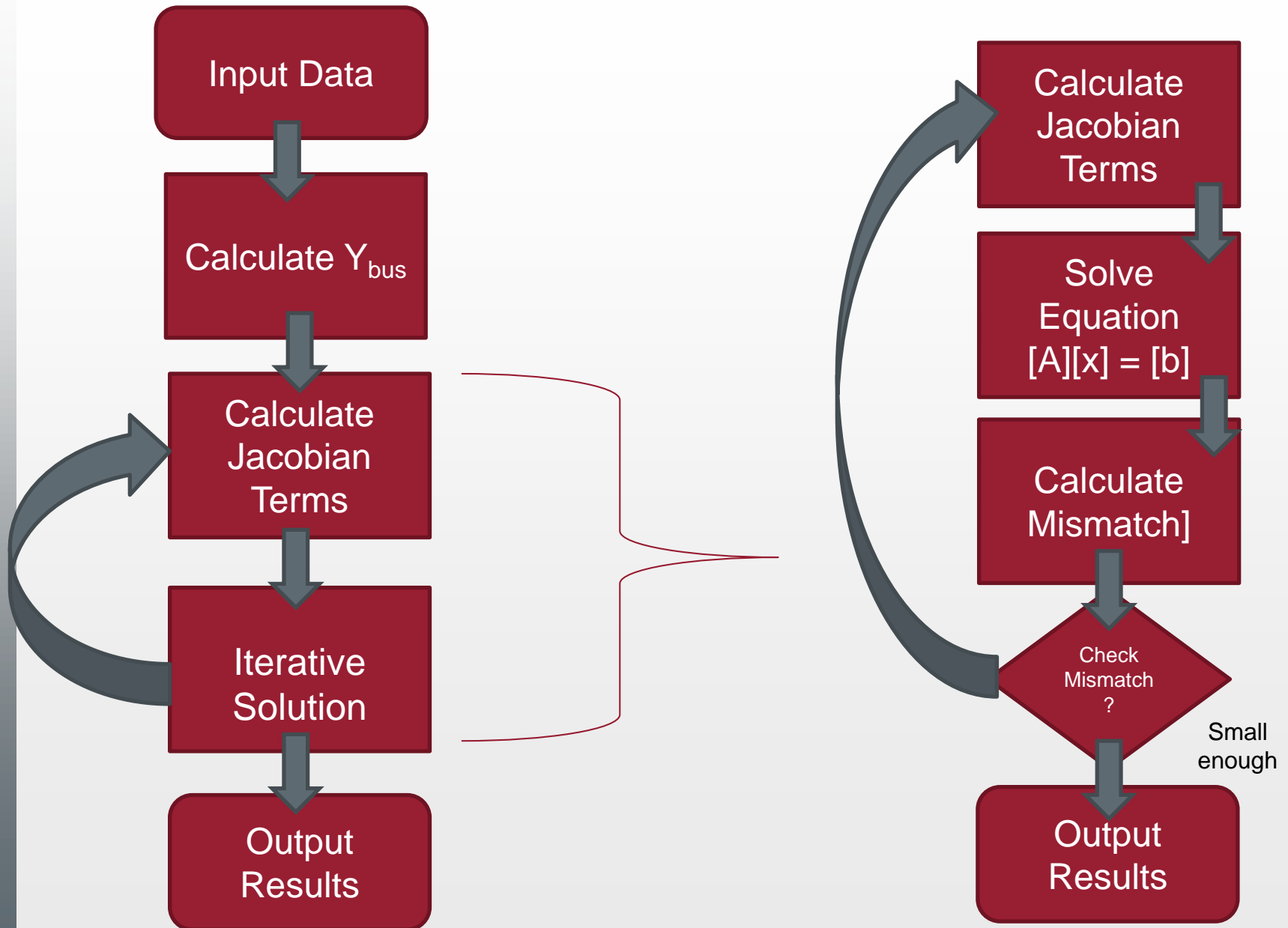
- [Amazon.com : matlab](#)
- [Learn with MATLAB and Simulink Tutorials - MATLAB & Simulink \(mathworks.com\)](#)
- [Matrices in the MATLAB Environment - MATLAB & Simulink \(mathworks.com\)](#)

# Programming Projects

## 1. Power Flow Project (20%)

- A.  $Y_{bus}$ , Jacobian matrices – Newton Raphson
- B. LU Factorization, Backward-Forward substitution for NR
- C. Fast Decoupled
- D. Q-limits, transformer taps

# Newton-Raphson and Parts



# **Math Methods behind solving Matrices**

## Stationary Iterative methods for solving $Ax=b$ :

Iterative methods generate a sequence of approximations to  $x^*$  from an initial guess (i.e.  $x^0, x^1, x^2, \dots x^k$ ) such that  $x^k \rightarrow x^*$  as  $k \rightarrow \infty$ .

If convergent<sup>1</sup>, these methods are accurate to within a user-defined convergence criterion.

Examples include:

1. Gauss Jacobi
2. Gauss Seidel
3. Successive Over-Relaxation

<sup>1</sup> we'll discuss convergence later

## Relaxation methods for solving $Ax=b$ :

Start with  $Ax = b$

Define a matrix  $M$  such that:  $Ax = b + Mx - Mx$

Rearranging:  $Mx = (M - A)x + b$

Adding iteration indices:  $Mx^{k+1} = (M - A)x^k + b \quad k = 1, \dots, \infty$

The choice of  $M$  determines if, and how fast,  $Ax - b \rightarrow 0$ .

When  $\|Ax^{(k+1)} - b\| < \varepsilon$ , for some small  $\varepsilon$ , then the iteration is said to have **converged**.



## Jacobi

Let  $M = D$  (the diagonal elements of  $A$ ), then

$$\begin{aligned} Dx^{k+1} &= (D - A)x^k + b \\ x^{k+1} &= -D^{-1}((A - D)x^k - b) \\ &= M_J x^k + D^{-1}b \end{aligned}$$

Or in scalar form: 
$$x_i^{k+1} = -\sum_{j \neq i}^n \left( \frac{a_{ij}}{a_{ii}} \right) x_j^k + \frac{b_i}{a_{ii}} \quad 1 \leq i \leq n, \quad k \geq 0$$

This method is attractive for parallel processing applications because each  $x_i^{k+1}$  depends only previous values of  $x^k$ . Thus each  $i^{th}$  element can be calculated independently.

Solve 
$$\begin{bmatrix} -10 & 2 & 3 & 6 \\ 0 & -9 & 1 & 4 \\ 2 & 6 & -12 & 2 \\ 3 & 1 & 0 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

using the Jacobi method.

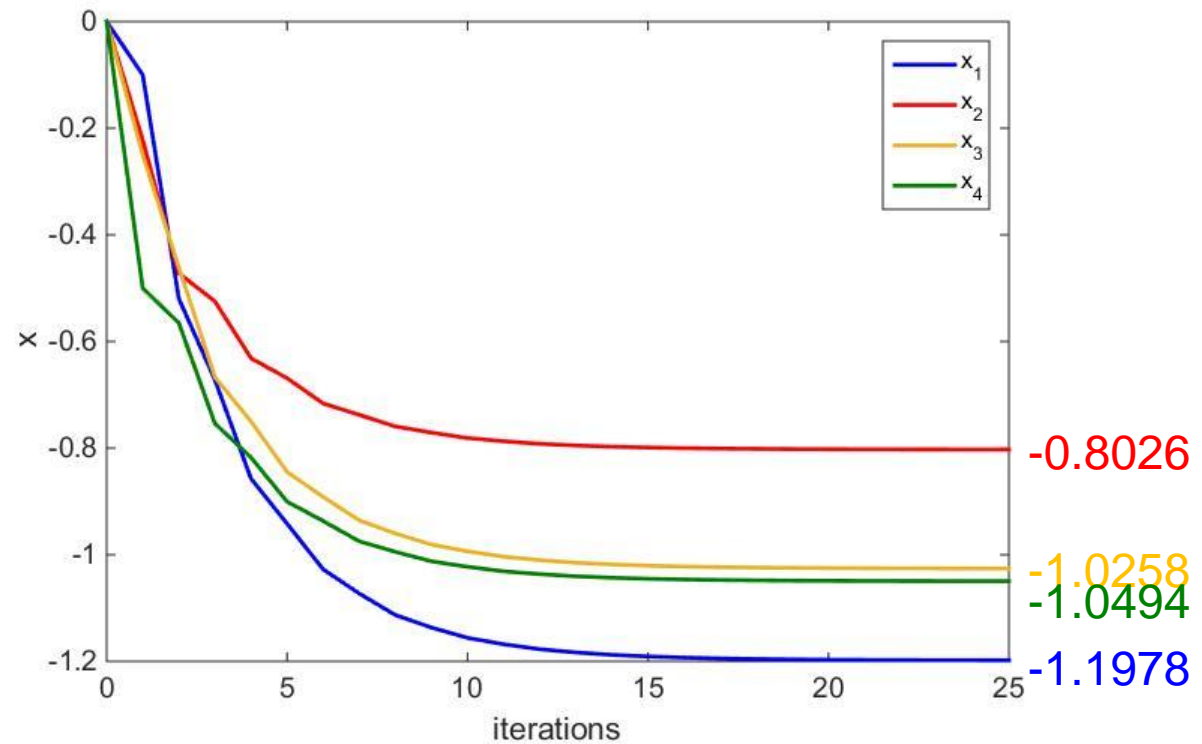
$$x_i^{k+1} = -\sum_{j \neq i}^n \left( \frac{a_{ij}}{a_{ii}} \right) x_j^k + \frac{b_i}{a_{ii}} \quad 1 \leq i \leq n, \quad k \geq 0$$

Set up the equations:

$$x_1^{k+1} = \frac{1}{10} (2x_2^k + 3x_3^k + 6x_4^k - 1)$$
$$x_2^{k+1} = \frac{1}{9} (0x_1^k + x_3^k + 4x_4^k - 2)$$
$$x_3^{k+1} = \frac{1}{12} (2x_1^k + 6x_2^k + 2x_4^k - 3)$$
$$x_4^{k+1} = \frac{1}{8} (3x_1^k + x_2^k + 0x_3^k - 4)$$

## Jacobi

$k$	$x_1$	$x_2$	$x_3$	$x_4$
1	0.0000	0.0000	0.0000	0.0000
2	-0.1000	-0.2222	-0.2500	-0.5000
3	-0.5194	-0.4722	-0.4611	-0.5653
4	-0.6719	-0.5247	-0.6669	-0.7538
5	-0.8573	-0.6314	-0.7500	-0.8176
6	-0.9418	-0.6689	-0.8448	-0.9004
7	-1.0275	-0.7163	-0.8915	-0.9368
8	-1.0728	-0.7376	-0.9355	-0.9748
9	-1.1131	-0.7594	-0.9601	-0.9945
10	-1.1366	-0.7709	-0.9810	-1.0123
11	-1.1559	-0.7811	-0.9936	-1.0226
12	-1.1679	-0.7871	-1.0037	-1.0311
13	-1.1772	-0.7920	-1.0100	-1.0363
14	-1.1832	-0.7950	-1.0149	-1.0404
15	-1.1877	-0.7974	-1.0181	-1.0431
16	-1.1908	-0.7989	-1.0205	-1.0451
17	-1.1930	-0.8001	-1.0221	-1.0464
18	-1.1945	-0.8009	-1.0233	-1.0474
19	-1.1956	-0.8014	-1.0241	-1.0480
20	-1.1963	-0.8018	-1.0247	-1.0485
21	-1.1969	-0.8021	-1.0250	-1.0489
22	-1.1972	-0.8023	-1.0253	-1.0491
23	-1.1975	-0.8024	-1.0255	-1.0492
24	-1.1977	-0.8025	-1.0257	-1.0494
25	-1.1978	-0.8026	-1.0258	-1.0494



Solve 
$$\begin{bmatrix} -10 & 2 & 3 & 6 \\ 0 & -9 & 1 & 4 \\ 2 & 6 & -12 & 2 \\ 3 & 1 & 0 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$
 using the Gauss-Seidel method.

$$x_i^{k+1} = -\sum_{j=1}^{i-1} \left( \frac{a_{ij}}{a_{ii}} \right) x_j^{k+1} - \sum_{j=i+1}^n \left( \frac{a_{ij}}{a_{ii}} \right) x_j^k + \frac{b_i}{a_{ii}} \quad 1 \leq i \leq n, \quad k \geq 0$$

Set up the equations:

$$x_1^{k+1} = \frac{1}{10} (2x_2^k + 3x_3^k + 6x_4^k - 1)$$

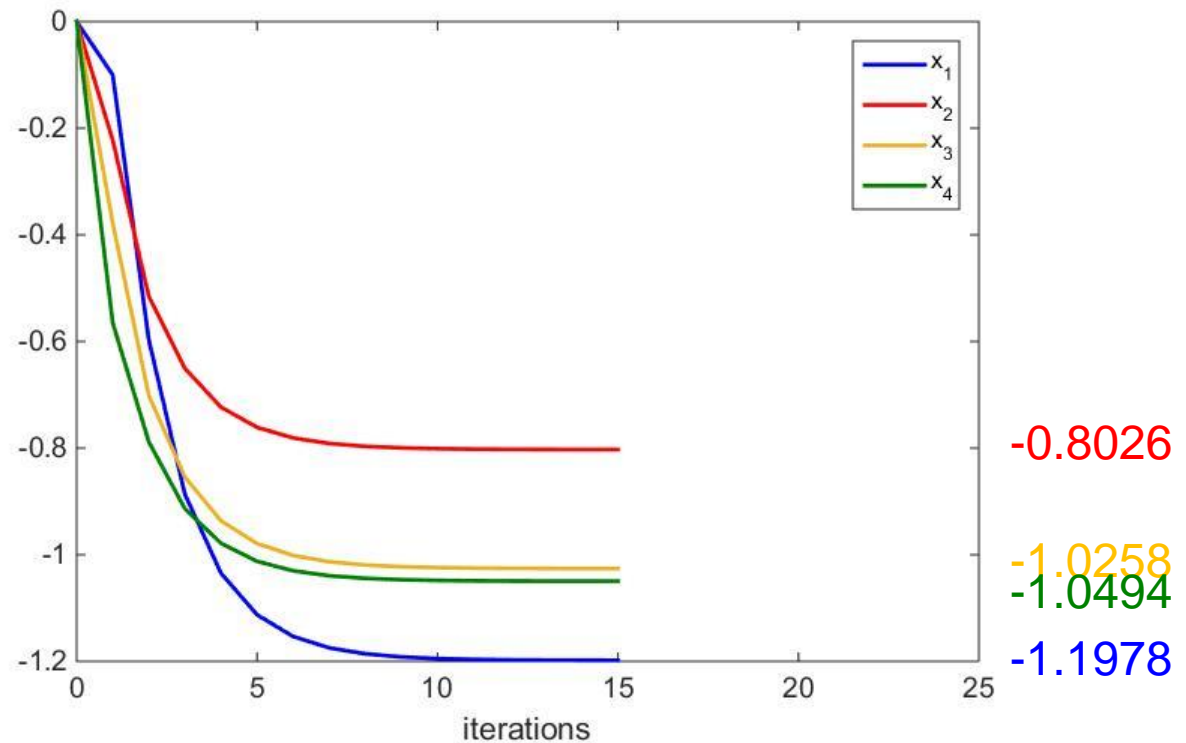
$$x_2^{k+1} = \frac{1}{9} (0x_1^{k+1} + x_3^k + 4x_4^k - 2)$$

$$x_3^{k+1} = \frac{1}{12} (2x_1^{k+1} + 6x_2^{k+1} + 2x_4^k - 3)$$

$$x_4^{k+1} = \frac{1}{8} (3x_1^{k+1} + x_2^{k+1} + 0x_3^{k+1} - 4)$$

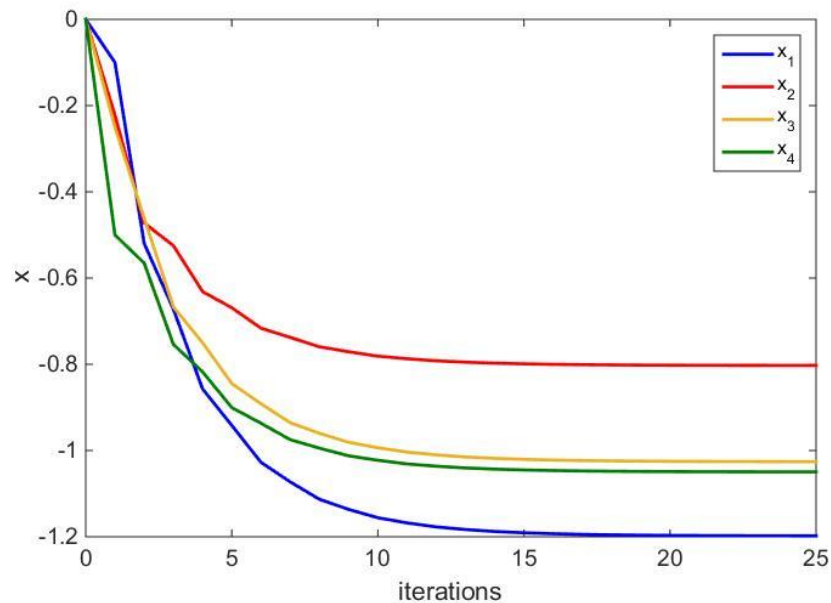
## Gauss-Seidel

$k$	$x_1$	$x_2$	$x_3$	$x_4$
1	0.0000	0.0000	0.0000	0.0000
2	-0.1000	-0.2222	-0.3778	-0.5653
3	-0.5969	-0.5154	-0.7014	-0.7883
4	-0.8865	-0.6505	-0.8544	-0.9137
5	-1.0347	-0.7233	-0.9364	-0.9784
6	-1.1126	-0.7611	-0.9791	-1.0124
7	-1.1534	-0.7809	-1.0014	-1.0301
8	-1.1747	-0.7913	-1.0131	-1.0394
9	-1.1859	-0.7968	-1.0193	-1.0443
10	-1.1917	-0.7996	-1.0225	-1.0468
11	-1.1948	-0.8011	-1.0241	-1.0482
12	-1.1964	-0.8019	-1.0250	-1.0489
13	-1.1972	-0.8023	-1.0255	-1.0492
14	-1.1976	-0.8025	-1.0257	-1.0494
15	-1.1979	-0.8026	-1.0259	-1.0495
16	-1.1980	-0.8027	-1.0259	-1.0496



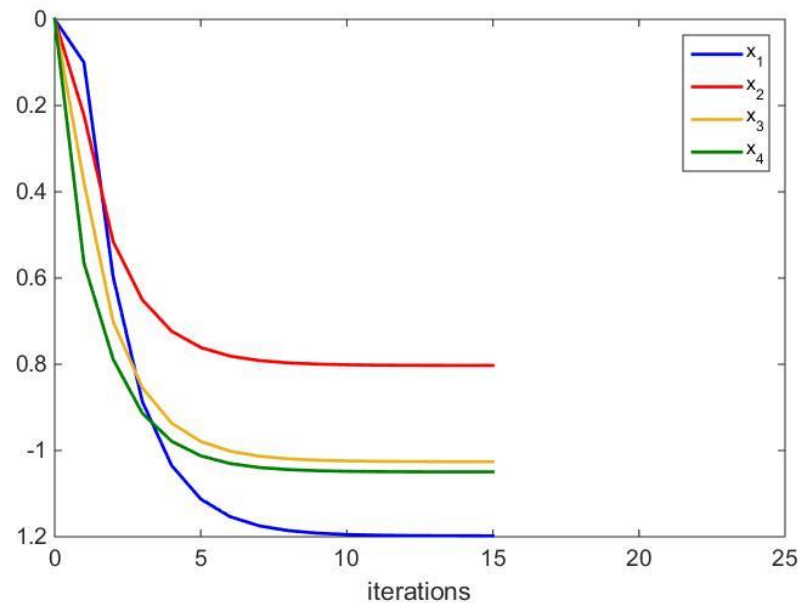
## Jacobi

$$x_i^{k+1} = -\sum_{j \neq i}^n \left( \frac{a_{ij}}{a_{ii}} \right) x_j^k + \frac{b_i}{a_{ii}} \quad 1 \leq i \leq n, \quad k \geq 0$$



## Gauss-Seidel

$$x_i^{k+1} = -\sum_{j=1}^{i-1} \left( \frac{a_{ij}}{a_{ii}} \right) x_j^{k+1} - \sum_{j=i+1}^n \left( \frac{a_{ij}}{a_{ii}} \right) x_j^k + \frac{b_i}{a_{ii}} \quad 1 \leq i \leq n, \quad k \geq 0$$



The Gauss Seidel method took fewer iterations to converge than did the Gauss Jacobi method.

Is it possible to predict if, and how fast, a method may converge?

Consider the matrices  $M_J$  and  $M_{GS}$ . If all of the eigenvalues of  $M_J$  or  $M_{GS}$  lie within the unit circle in the complex plane, then the iteration will converge for any initial guess.

$$M_J \triangleq -D^{-1} (L + U) \qquad M_{GS} \triangleq - (L + D)^{-1} U$$

$$M_J = \begin{bmatrix} 0 & -0.2000 & -0.3000 & -0.6000 \\ 0 & 0 & -0.1111 & -0.4444 \\ -0.1667 & -0.5000 & 0 & -0.1667 \\ -0.3750 & -0.1250 & 0 & 0 \end{bmatrix} \longrightarrow \text{eig}(M_J) = \begin{bmatrix} -0.7029 \\ 0.5679 \\ 0.0675 + j0.1636 \\ 0.0675 - j0.1636 \end{bmatrix} < \text{unit circle}$$

Convergence guaranteed!

$$M_{GS} = \begin{bmatrix} 0 & -0.2000 & -0.3000 & -0.6000 \\ 0 & 0 & -0.1111 & -0.4444 \\ 0 & -0.0333 & -0.1056 & -0.4889 \\ 0 & -0.0750 & -0.1264 & -0.2806 \end{bmatrix} \longrightarrow \text{eig}(M_{GS}) = \begin{bmatrix} 0 \\ -0.5234 \\ 0.0233 \\ 0.1140 \end{bmatrix} < \text{unit circle}$$

Convergence guaranteed!



Consider now the Jacobi method applied to the following system of equations:

$$\begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & 1 & 2 & 3 \\ 4 & 3 & 5 & 8 \\ 9 & 2 & 7 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad (\text{used previously in LU factorization example})$$

$$M_J = -D^{-1}(L+U)$$

$$= \begin{bmatrix} 0.00 & -3.00 & -4.00 & -8.00 \\ -2.00 & 0.00 & -2.00 & -3.00 \\ -0.80 & -0.60 & -0.00 & -1.60 \\ -2.25 & -0.50 & -1.75 & -0.00 \end{bmatrix} \quad eig(M_J) = \begin{bmatrix} -6.6212 \\ 4.3574 \\ 1.2072 \\ 1.0566 \end{bmatrix} > \text{unit circle}$$

Iterations will not converge

$k$	$x_1$	$x_2$	$x_3$	$x_4$
1	0	0	0	0
2	1.0000	1.0000	0.2000	0.2500
3	-4.8000	-2.1500	-1.6000	-2.8500
4	36.6500	22.3500	9.8900	14.9250
5	-225.0100	-136.8550	-66.4100	-110.6950

Not converging!

$$\text{eig}(M_J) = \begin{bmatrix} -6.6212 \\ 4.3574 \\ 1.2072 \\ 1.0566 \end{bmatrix} > \text{unit circle} \quad \text{Iterations will not converge}$$


$k$	$x_1$	$x_2$	$x_3$	$x_4$
1	0	0	0	0
2	1.0000	1.0000	0.2000	0.2500
3	-4.8000	-2.1500	-1.6000	-2.8500
4	36.6500	22.3500	9.8900	14.9250
5	-225.0100	-136.8550	-66.4100	-110.6950

Not converging!

## Successive Overrelaxation

The Gauss-Seidel iteration  $x^{k+1} = -(L + D)^{-1}((A - L - D)x^k - b)$

can be rewritten as:  $x^{k+1} = x^k - D^{-1}(Lx^{k+1} + (D + U)x^k - b)$  Upper triangular part of A  
(not the U from LU  
factorization)



Add a weighting factor  $0 < \omega < 2$ :  $x^{k+1} = x^k - \omega D^{-1}(Lx^{k+1} + (D + U)x^k - b)$

If  $\omega = 1$ , then the method reduces to the Gauss-Seidel.

For large systems, this method can potentially converge much more rapidly than either Jacobi or Gauss-Seidel, but it is difficult to find an optimal value of  $\omega$

## Subspace methods

The classical iterative solvers discussed up to this point were of the form

$$x^{k+1} = Gx^k + c$$

with constant  $G$  and  $c$ . Such methods are also known as *stationary methods*. We will now study a different class of iterative solvers based on optimization.

These methods typically attempt to minimize the error in the residual of  $Ax=b$  at every step by successively reducing the error in the *subspace* defined by carefully selected vectors.

Subspace methods do not use an iterative matrix like stationary methods, but rather update the current approximation based on past vector information.

Additionally, unlike stationary methods, subspace methods are typically guaranteed to converge in **at most  $n$**  iterations. Some methods require special matrix features such as positive definiteness.

These methods may converge rapidly if the  $A$  matrix is very large with relatively few non-zero entries.

## Conjugate Gradient Methods

The Conjugate Gradient (CG) method requires that  $A$  be positive definite, i.e. that  $x^T A x > 0$  for any non-zero vector  $x$ .

The CG method can be considered a minimization method for the error function

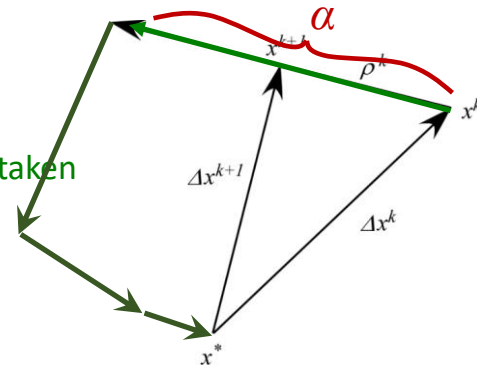
$$E(x) = \|Ax - b\|^2$$

along a succession of directional vectors (rays).

The CG method produces a series of vector iterates, such that

$$x^{k+1} = x^k + \alpha_{k+1} \rho_{k+1}$$

step length (i.e. how big a step is taken)  
direction in which step is taken



To eventually reach  $x^*$  in a finite number of steps

## Conjugate Gradient algorithm for solving $Ax=b$

Initialization: Let  $k=1$ , and

$$r_0 = b - Ax^0 \quad \rho_0 = \|r_0\|^2$$

While  $\|r_k\| \geq \varepsilon$    $r_k$  is called the “residual”

if  $k=1$ ,  $\sigma = r_0$ , otherwise  $\sigma = r_{k-1} + \beta\sigma$ , where  $\beta = \frac{\rho_{k-1}}{\rho_{k-2}}$

$$w = A\sigma$$

$$\alpha = \frac{\rho_{k-1}}{\sigma^T w}$$

$$x = x + \alpha\sigma$$

$$r_k = r_{k-1} - \alpha w$$

$$\rho_k = \|r_k\|^2$$

$$k = k + 1$$



Solve

$$\begin{bmatrix} 22 & 2 & 8 & 5 & 3 \\ 2 & 19 & 6 & 6 & 2 \\ 8 & 6 & 27 & 7 & 4 \\ 5 & 6 & 7 & 24 & 0 \\ 3 & 2 & 4 & 0 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

using the Conjugate Gradient  
method with zeros as the initial  
conditions

Initialize:

$$r_0 = b - Ax^0 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

$$\rho_0 = \|r_0\|^2 = 55$$

$$k = 1$$

$$\text{since } k = 1, \sigma = r_0 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

$$w = A\sigma = \begin{bmatrix} 85 \\ 92 \\ 149 \\ 134 \\ 64 \end{bmatrix}$$

$$\alpha = \frac{\rho_0}{\sigma^T w} = 0.0350$$

$$x = x + \alpha\sigma = \begin{bmatrix} 0.0350 \\ 0.0700 \\ 0.1050 \\ 0.1399 \\ 0.1749 \end{bmatrix}$$

$$r_1 = r_0 - \alpha w = \begin{bmatrix} -1.9739 \\ -1.2188 \\ -2.2131 \\ -0.6883 \\ 2.7608 \end{bmatrix}$$

$$\rho_1 = \|r_1\|^2 = 18.3756$$

 Watch this number

$$k = 2 \quad \beta = \frac{\rho_1}{\rho_0} = 0.3341$$

$$\sigma = r_1 + \beta \sigma = \begin{bmatrix} -1.6398 \\ -0.5506 \\ -1.2108 \\ 0.6481 \\ 4.4313 \end{bmatrix}$$

$$w = A\sigma = \begin{bmatrix} -30.3291 \\ -8.2550 \\ -26.8518 \\ -4.4238 \\ 29.0180 \end{bmatrix}$$

$$\alpha = \frac{\rho_1}{\sigma^T w} = 0.0865$$

$$x = x + \alpha \sigma = \begin{bmatrix} -0.1068 \\ 0.0224 \\ 0.0003 \\ 0.1960 \\ 0.5581 \end{bmatrix}$$

$$r_2 = r_1 - \alpha w = \begin{bmatrix} 0.6486 \\ -0.5050 \\ 0.1087 \\ -0.3058 \\ 0.2517 \end{bmatrix}$$

$$\rho_2 = \|r_2\|^2 = 0.8444$$

$$k = 3 \quad \beta = \frac{\rho_2}{\rho_1} = 0.0460$$

$$\sigma = r_2 + \beta \sigma = \begin{bmatrix} 0.5732 \\ -0.5303 \\ 0.0531 \\ -0.2760 \\ 0.4553 \end{bmatrix}$$

$$w = A\sigma = \begin{bmatrix} 11.9610 \\ -9.3567 \\ 2.7264 \\ -6.5682 \\ 4.9692 \end{bmatrix}$$

$$\alpha = \frac{\rho_2}{\sigma^T w} = 0.0526$$

$$x = x + \alpha \sigma = \begin{bmatrix} -0.0766 \\ -0.0056 \\ 0.0031 \\ 0.1815 \\ 0.5821 \end{bmatrix}$$

$$r_3 = r_2 - \alpha w = \begin{bmatrix} 0.0189 \\ -0.0124 \\ -0.0348 \\ 0.0400 \\ -0.0099 \end{bmatrix}$$

$$\rho_3 = \|r_3\|^2 = 0.0034$$

$$k = 4 \quad \beta = \frac{\rho_3}{\rho_2} = 0.0041$$

$$\sigma = r_3 + \beta \sigma = \begin{bmatrix} 0.0212 \\ -0.0146 \\ -0.0346 \\ 0.0389 \\ -0.0081 \end{bmatrix}$$

$$w = A\sigma = \begin{bmatrix} 0.3306 \\ -0.2250 \\ -0.6121 \\ 0.7098 \\ -0.1767 \end{bmatrix}$$

$$\alpha = \frac{\rho_3}{\sigma^T w} = 0.0566$$

$$x = x + \alpha \sigma = \begin{bmatrix} -0.0754 \\ -0.0064 \\ 0.0011 \\ 0.1837 \\ 0.5816 \end{bmatrix}$$

$$r_4 = r_3 - \alpha w = \begin{bmatrix} 0.1720 \\ 0.2970 \\ -0.1888 \\ -0.1360 \\ 0.0689 \end{bmatrix} \times 10^{-3}$$

$$\rho_4 = \|r_4\|^2 = 1.7667 \times 10^{-7}$$

This is sufficiently small, therefore the method has converged in less than 5 iterations!

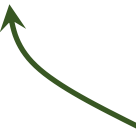
If the  $A$  matrix is not symmetric or positive definite, the conjugate gradient method is not guaranteed to work.

(it may still converge, but probably with poor convergence)

An alternate method is the Generalized Minimal Residual Algorithm (GMRES)

The  $k^{\text{th}}$  iteration of the GMRES method is the solution to the least squares problem

$$\min_{x \in x_0 + K_k} \|b - Ax\|$$



the Krylov subspace

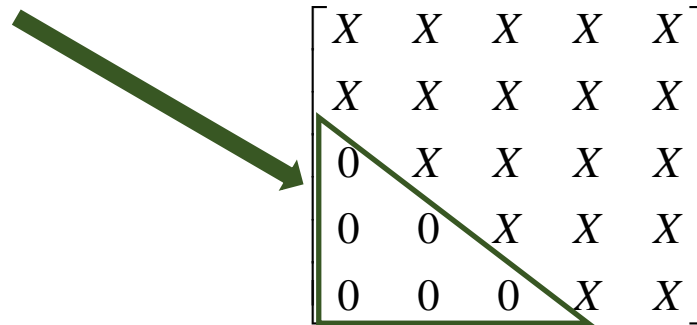
Some necessary precursors:

One of the components of the GMRES method is based on similarity transformations of the form

$$A = QHQ^*$$

where  $H$  is an upper Hessenberg matrix.

An **upper Hessenberg matrix** is a square matrix that has zero entries below the first subdiagonal.



A Given's rotation is used to remove single nonzero elements of matrices in reduction to triangular (Hessenberg) form.

The Given's rotation to zero out element  $A_{jk}$  is a transformation based on the matrix

$$G_{jk} = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & & \vdots & & & & \vdots \\ 0 & \dots & cs & \dots & sn & \dots & 0 \\ \vdots & & \vdots & & & & \vdots \\ 0 & \dots & -sn & \dots & cs & \dots & 0 \\ \vdots & & \vdots & & & & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix} \begin{matrix} \\ \\ j^{th} \text{ row} \\ \\ k^{th} \text{ row} \\ \\ \end{matrix}$$

$j^{th} \text{ column} \quad k^{th} \text{ column}$

$cs = \cos(\phi)$   
 $sn = \sin(\phi)$



## GMRES algorithm for solving $Ax=b$

Initialize:  $r_0 = b - Ax^0$

$$e_1 = [1 \ 0 \ 0 \ \dots \ 0]^T$$

$$v_1 = \frac{r_0}{\|r_0\|}$$

$$s = \|r_0\|e_1$$

$$k = 1$$

$$cs = [0 \ 0 \ 0 \ \dots \ 0]^T$$

$$sn = [0 \ 0 \ 0 \ \dots \ 0]^T$$

While  $\|r_k\| \geq \varepsilon$  and  $k \leq k_{\max}$ , set

$$1. H(j, k) = (Av_k)^T v_j, \quad j = 1, \dots, k$$

$$2. v_{k+1} = Av_k - \sum_{j=1}^k H(j, k)v_j$$

$$3. H(k+1, k) = \|v_{k+1}\|$$

$$4. v_{k+1} = \frac{v_{k+1}}{\|v_{k+1}\|}$$

### 5. Given's Rotation :

$$(a) \quad \begin{bmatrix} H(j,k) \\ H(j+1,k) \end{bmatrix} = \begin{bmatrix} cs(j) & sn(j) \\ -sn(j) & cs(j) \end{bmatrix} \begin{bmatrix} H(j,k) \\ H(j+1,k) \end{bmatrix}, \quad j = 1, \dots, k-1$$

$$(b) \quad cs(k) = \frac{H(k,k)}{\sqrt{H(k+1,k)^2 + H(k,k)^2}}, \quad sn(k) = \frac{H(k+1,k)}{\sqrt{H(k+1,k)^2 + H(k,k)^2}}$$

### (c) Approximate residual norm

$$\alpha = cs(k)s(k)$$

$$s(k+1) = -sn(k)s(k)$$

$$s(k) = \alpha$$

$$\text{error} = |s(k+1)|$$

### (d) Set

$$H(k,k) = cs(k)H(k,k) + sn(k)H(k+1,k)$$

$$H(k+1,k) = 0$$

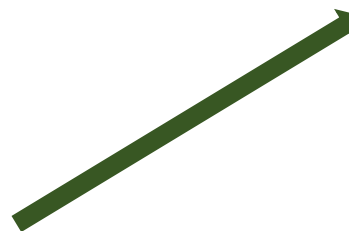
### 6. If error $\leq \varepsilon$

(a) Solve  $Hy = s$  for  $y$

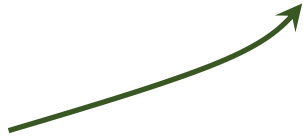
(b) Calculate  $x = x - Vy$

(c) Method has converged. Return.

Otherwise  $k = k + 1$ , repeat.



Solve 
$$\begin{bmatrix} -10 & 2 & 3 & 6 \\ 0 & -9 & 1 & 4 \\ 2 & 6 & -12 & 2 \\ 3 & 1 & 0 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$
 using the GMRES method with zeros as the initial conditions and  $\varepsilon = 10^{-3}$

Convergence tolerance 

Initialize:

$$r = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad v = \begin{bmatrix} 0.1826 \\ 0.3651 \\ 0.5477 \\ 0.7303 \end{bmatrix}, \quad s = \begin{bmatrix} 5.4772 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$k = 1$$

$$H(j,1) = (Av_1)^T v_j, \quad j=1$$

$$= \begin{bmatrix} -4.0333 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$v_2 = Av_1 - \sum_{j=1}^1 H(j,1)v_j$$

$$= \begin{bmatrix} 5.6659 \\ 1.6553 \\ -0.3469 \\ -1.9840 \end{bmatrix}$$

$$H(2,1) = \|v_2\|$$

$$= \begin{bmatrix} -4.0333 & 0 & 0 & 0 \\ 6.2369 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$v_2 = \frac{v_2}{\|v_2\|}$$

$$= \begin{bmatrix} 0.9084 \\ 0.2654 \\ -0.0556 \\ -0.3181 \end{bmatrix}$$

Applying Given's rotation to zero out  $H(2,1)$ :

5. Given's Rotation :

$$(a) \begin{bmatrix} H(j,k) \\ H(j+1,k) \end{bmatrix} = \begin{bmatrix} cs(j) & sn(j) \\ -sn(j) & cs(j) \end{bmatrix} \begin{bmatrix} H(j,k) \\ H(j+1,k) \end{bmatrix}, \quad j = 1, \dots, k-1$$

Since  $k = 1$ , this step isn't performed



$$(b) \quad cs(1) = \frac{H(1,1)}{\sqrt{H(2,1)^2 + H(1,1)^2}} = -0.5430$$

$$sn(1) = \frac{H(2,1)}{\sqrt{H(2,1)^2 + H(1,1)^2}} = 0.8397$$

(c) Approximate residual norm

$$\alpha = cs(1)s(1) = -2.9743$$

$$s(2) = -sn(1)s(1) = -4.5993$$

$$s(1) = \alpha \Rightarrow s = \begin{bmatrix} -2.9743 \\ -4.5993 \\ 0 \\ 0 \end{bmatrix}$$

$$\text{error} = |s(2)| = 4.5993$$

(d) Set

$$H(1,1) = cs(1)H(1,1) + sn(1)H(2,1) = 7.4274$$

$$H(2,1) = 0$$

$$H = \begin{bmatrix} 7.4274 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Now zero

Since error (= 4.5993) is greater than  $\varepsilon$ , repeat process

$$k = 2$$

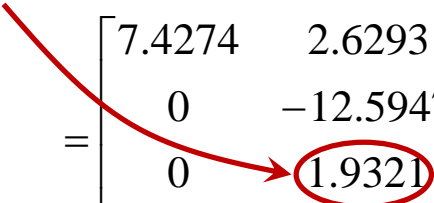
$$H(j,2) = (Av_2)^T v_j, \quad j = 1, 2$$

$$= \begin{bmatrix} 7.4274 & 2.6293 & 0 & 0 \\ 0 & -12.5947 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$v_3 = Av_2 - \sum_{j=1}^2 H(j,2)v_j$$

$$= \begin{bmatrix} 0.3324 \\ -1.3341 \\ 1.2999 \\ -0.3910 \end{bmatrix}$$

$$H(3,2) = \|v_3\|$$

$$= \begin{bmatrix} 7.4274 & 2.6293 & 0 & 0 \\ 0 & -12.5947 & 0 & 0 \\ 0 & 1.9321 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$


$$v_3 = \frac{v_3}{\|v_3\|}$$

$$= \begin{bmatrix} 0.1721 \\ -0.6905 \\ 0.6728 \\ -0.2024 \end{bmatrix}$$

Applying Given's rotation to zero out  $H(3,2)$ :

5. Given's Rotation :

$$(a) \quad \begin{bmatrix} H(1,2) \\ H(2,2) \end{bmatrix} = \begin{bmatrix} cs(1) & sn(1) \\ -sn(1) & cs(1) \end{bmatrix} \begin{bmatrix} H(1,2) \\ H(2,2) \end{bmatrix} \\ = \begin{bmatrix} -12.0037 \\ 4.6314 \end{bmatrix}$$

$$(b) \quad cs(2) = \frac{H(2,2)}{\sqrt{H(3,2)^2 + H(2,2)^2}} = 0.9229 \\ sn(2) = \frac{H(3,2)}{\sqrt{H(3,2)^2 + H(2,2)^2}} = 0.3850$$



(c) Approximate residual norm

$$\alpha = cs(2)s(2) = -4.2447$$

$$s(3) = -sn(2)s(2) = 1.7708$$

$$s(2) = \alpha \Rightarrow s = \begin{bmatrix} -2.9743 \\ -4.2447 \\ 1.7708 \\ 0 \end{bmatrix}$$

$$\text{error} = |s(3)| = 1.7708$$

(d) Set

$$H(2,2) = cs(2)H(2,2) + sn(2)H(3,2) = 5.0183$$

$$H(3,2) = 0$$

$$H = \begin{bmatrix} 7.4274 & -12.0037 & 0 & 0 \\ 0 & 5.0183 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Now zero

Since error (= 1.7708) is greater than  $\varepsilon$ , repeat process

$$k = 3$$

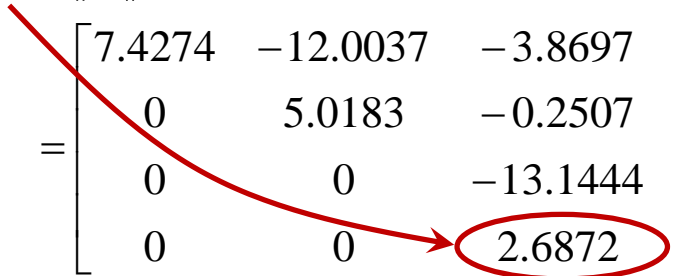
$$H(j,3) = (Av_3)^T v_j, \quad j = 1, 2, 3$$

$$= \begin{bmatrix} 7.4274 & -12.0037 & -3.8697 & 0 \\ 0 & 5.0183 & -0.2507 & 0 \\ 0 & 0 & -13.1444 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$v_4 = Av_3 - \sum_{j=1}^3 H(j,3)v_j$$

$$= \begin{bmatrix} 0.3343 \\ -0.5652 \\ -0.4942 \\ 0.5697 \end{bmatrix}$$

$$H(4,3) = \|v_4\|$$

$$= \begin{bmatrix} 7.4274 & -12.0037 & -3.8697 & 0 \\ 0 & 5.0183 & -0.2507 & 0 \\ 0 & 0 & -13.1444 & 0 \\ 0 & 0 & 2.6872 & 0 \end{bmatrix}$$


$$v_4 = \frac{v_4}{\|v_4\|}$$

$$= \begin{bmatrix} 0.3343 \\ -0.5652 \\ -0.4942 \\ 0.5697 \end{bmatrix}$$

Applying Given's rotation to zero out  $H(4,3)$ :

5. Given's Rotation :

$$(a) \begin{bmatrix} H(1,3) \\ H(2,3) \end{bmatrix} = \begin{bmatrix} cs(1) & sn(1) \\ -sn(1) & cs(1) \end{bmatrix} \begin{bmatrix} H(1,3) \\ H(2,3) \end{bmatrix}$$

$$= \begin{bmatrix} 1.8908 \\ 3.3855 \end{bmatrix}$$

$$\begin{bmatrix} H(2,3) \\ H(3,3) \end{bmatrix} = \begin{bmatrix} cs(2) & sn(2) \\ -sn(2) & cs(2) \end{bmatrix} \begin{bmatrix} H(2,3) \\ H(3,3) \end{bmatrix}$$

$$= \begin{bmatrix} -1.9362 \\ -13.4346 \end{bmatrix}$$

Note: for  $j = 1$  and 2

$$(b) \quad cs(3) = \frac{H(3,3)}{\sqrt{H(4,3)^2 + H(3,3)^2}} = -0.9806$$

$$sn(3) = \frac{H(4,3)}{\sqrt{H(4,3)^2 + H(4,3)^2}} = 0.1961$$

(c) Approximate residual norm

$$\alpha = cs(3)s(3) = -1.7364$$

$$s(4) = -sn(3)s(3) = -0.3473$$

$$s(3) = \alpha \Rightarrow s = \begin{bmatrix} -2.9743 \\ -4.2447 \\ -1.7364 \\ -0.3473 \end{bmatrix}$$

$$\text{error} = |s(4)| = 0.3473$$

(d) Set

$$H(3,3) = cs(3)H(3,3) + sn(3)H(4,3) = 13.7007$$

$$H(4,3) = 0$$

$$H = \begin{bmatrix} 7.4274 & -12.0037 & 1.8908 & 0 \\ 0 & 5.0183 & -1.9362 & 0 \\ 0 & 0 & 13.7007 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Now zero

Since error (= 0.3473) is greater than  $\varepsilon$ , repeat process

$$k = 4$$

$$H(j,4) = (Av_4)^T v_j, \quad j = 1, 2, 3, 4$$

$$= \begin{bmatrix} 7.4274 & -12.0037 & 1.8908 & 1.4182 \\ 0 & 5.0183 & -1.9362 & 0.5863 \\ 0 & 0 & 13.7007 & -1.4228 \\ 0 & 0 & 0 & -9.2276 \end{bmatrix}$$

$$v_5 = Av_4 - \sum_{j=1}^4 H(j,4)v_j$$



Don't need to perform this step since  $n = 4$

## Applying Given's rotation

### 5. Given's Rotation :

$$(a) \begin{bmatrix} H(1,4) \\ H(2,4) \end{bmatrix} = \begin{bmatrix} cs(1) & sn(1) \\ -sn(1) & cs(1) \end{bmatrix} \begin{bmatrix} H(1,4) \\ H(2,4) \end{bmatrix}$$

$$= \begin{bmatrix} -1.5092 \end{bmatrix}$$

$$\begin{bmatrix} H(2,4) \\ H(3,4) \end{bmatrix} = \begin{bmatrix} cs(2) & sn(2) \\ -sn(2) & cs(2) \end{bmatrix} \begin{bmatrix} H(2,4) \\ H(3,4) \end{bmatrix}$$

$$= \begin{bmatrix} -1.9407 \\ -0.7320 \end{bmatrix}$$

Note: for  $j = 1, 2,$  and  $3$

$$\begin{bmatrix} H(3,4) \\ H(4,4) \end{bmatrix} = \begin{bmatrix} cs(3) & sn(3) \\ -sn(3) & cs(3) \end{bmatrix} \begin{bmatrix} H(3,4) \\ H(4,4) \end{bmatrix}$$

$$= \begin{bmatrix} -1.0920 \\ 9.1919 \end{bmatrix}$$

$$(b) \quad cs(4) = \frac{H(4,4)}{\sqrt{0 + H(4,4)^2}} = 1$$

$$sn(4) = \frac{0}{\sqrt{0 + H(4,4)^2}} = 0$$

(c) Approximate residual norm

$$\alpha = cs(4)s(4) = -0.3473$$

$$s(5) = -sn(4)s(4) = 0$$

$$s(4) = \alpha \Rightarrow s = \begin{bmatrix} -2.9743 \\ -4.2447 \\ -1.7364 \\ -0.3473 \end{bmatrix}$$

$$\text{error} = |s(5)| = 0$$

Since error is now less than  $\varepsilon$ ,

(a) Solve  $Hy = s$  for  $y$

$$y = \begin{bmatrix} -1.8404 \\ -0.9105 \\ -0.1297 \\ -0.0378 \end{bmatrix}$$

(b) Calculate  $x = x - Vy$

$$x = \begin{bmatrix} -1.1981 \\ -0.8027 \\ -1.0260 \\ -1.0496 \end{bmatrix}$$

(c) Method has converged.

## Preconditioners for Iterative Solvers

The rate of convergence of iterative methods depends on the condition number of the iteration matrix. Therefore, it may be advantageous to transform the linear system of equations into an equivalent system (i.e. one that has the same solution) that has more favorable properties.

The matrix, or matrices, that perform this transformation are called *preconditioners*.

$$M^{-1}Ax = M^{-1}b$$

If  $M^{-1}A$  approximates the identity matrix  $I$ , then the condition number is close to one and the iterates will converge rapidly



## Jacobi Preconditioner

In this preconditioning, the matrix  $M$  is chosen such that  $M = D$  (the diagonal elements of  $A$ ). The advantage of this method is that  $D$  is easy to find and  $D^{-1}$  is computationally efficient to calculate.

$$M = D$$

## Symmetric Successive Over-Relaxation Preconditioner

In this preconditioning, the matrix  $M$  uses the upper ( $U$ ) and lower ( $L$ ) triangular matrices of  $A$ .

$$M = \frac{1}{\omega(2-\omega)}(D + \omega L)D^{-1}(D + \omega U)$$

The factor  $\frac{1}{\omega(2-\omega)}$  has no impact on the convergence and can be neglected.

If  $\omega$  is taken to be equal to 1, then this preconditioner is called the *symmetric Gauss-Seidel* method.

## Incomplete LU factorization

For an effective preconditioner, it is desired to find a matrix  $M^{-1}$  that closely approximates  $A^{-1}$ .

Since  $U^{-1}L^{-1} = A^{-1}$  then approximations to  $U$  and  $L$  may lead to a good approximation to  $M^{-1}$  and  $U^{-1}$  and  $L^{-1}$  are computationally easy to compute (because they are triangular).

An LU factorization is called *incomplete* if, during the factorization process, one or more fill elements are ignored.

$$\text{Solve } \begin{bmatrix} 10 & 1 & 0 & 3 & 0 & 0 & 0 & 5 & 0 & 0 \\ 2 & 9 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 2 \\ 0 & 0 & 21 & 5 & 7 & 0 & 0 & 0 & 0 & 4 \\ 4 & 0 & 1 & 18 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 7 & 25 & 4 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 3 & 14 & 9 & 0 & 0 & 0 \\ 0 & 1 & 4 & 0 & 2 & 3 & 12 & 1 & 1 & 0 \\ 1 & 0 & 5 & 0 & 0 & 0 & 5 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 20 & 0 \\ 0 & 2 & 3 & 0 & 4 & 0 & 0 & 0 & 0 & 35 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{with different preconditioners}$$

using the GMES method

## Jacobi Preconditioner

$$M = D$$

$$M = D =$$

## Symmetric Successive Over-Relaxation Preconditioner

$$M = (D + \omega L)D^{-1}(D + \omega U) \quad \omega = 1$$

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 0 & 2 & 3 & 0 & 0 & 0 & 0 \\ 1 & 0 & 5 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 2 & 3 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & 1 & 0 & 3 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 2 \\ 0 & 0 & 0 & 5 & 7 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$M = \begin{bmatrix} 10 & 1 & 0 & 3 & 0 & 0 & 0 & 5 & 0 & 0 \\ 2 & 9.2 & 0 & 0.6 & 0 & 0 & 5 & 1 & 0 & 2 \\ 0 & 0 & 21 & 5 & 7 & 0 & 0 & 0 & 0 & 4 \\ 4 & 0.4 & 1 & 19.44 & 8.33 & 0 & 0 & 2 & 0 & 0.19 \\ 0 & 0 & 4 & 7.95 & 29.44 & 4 & 1 & 0 & 0 & 2.76 \\ 0 & 0 & 0 & 0 & 3 & 14.48 & 9.12 & 0 & 0 & 0.24 \\ 0 & 1 & 4 & 0.95 & 3.33 & 3.32 & 14.56 & 1 & 1 & 1.14 \\ 1 & 0.1 & 5 & 1.49 & 1.67 & 0 & 5 & 10.92 & 0.42 & 0.95 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0.50 & 20.5 & 0 \\ 0 & 2 & 3 & 0.71 & 5 & 0.64 & 1.27 & 0 & 0 & 36.34 \end{bmatrix}$$

## Incomplete LU factorization

$$L = \begin{bmatrix} 10.0000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2.0000 & 8.8000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 21.0000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4.0000 & 0 & 1.0000 & 16.5619 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4.0000 & 6.0476 & 20.8672 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.0000 & 13.4249 & 0 & 0 & 0 & 0 \\ 0 & 1.0000 & 4.0000 & 0 & 0.6667 & 2.8722 & 9.5051 & 0 & 0 & 0 \\ 1.0000 & 0 & 5.0000 & 0 & 0 & 0 & 5.0000 & 8.9740 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.0000 & 0 & 19.3688 & 0 \\ 0 & 2.0000 & 3.0000 & 0 & 3.0000 & 0 & 0 & 0 & 0 & 33.7960 \end{bmatrix}$$

$$U = \begin{bmatrix} 1.0000 & 0.1000 & 0 & 0.3000 & 0 & 0 & 0 & 0.5000 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 & 0 & 0 & 0.5682 & 0 & 0 & 0.2273 \\ 0 & 0 & 1.0000 & 0.2381 & 0.3333 & 0 & 0 & 0 & 0 & 0.1905 \\ 0 & 0 & 0 & 1.0000 & 0.4629 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 & 0.1917 & 0.0479 & 0 & 0 & 0.0593 \\ 0 & 0 & 0 & 0 & 0 & 1.0000 & 0.6597 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.0000 & 0.1052 & 0.1052 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

Note that these L and U matrices are different than the L and U matrices of the SSOR preconditioner

$$M = \begin{bmatrix} 10.0000 & 1.0000 & 0 & 3.0000 & 0 & 0 & 0 & 5.0000 & 0 & 0 \\ 2.0000 & 9.0000 & 0 & 0.6000 & 0 & 0 & 5.0000 & 1.0000 & 0 & 2.0000 \\ 0 & 0 & 21.0000 & 5.0000 & 7.0000 & 0 & 0 & 0 & 0 & 4.0000 \\ 4.0000 & 0.4000 & 1.0000 & 18.0000 & 8.0000 & 0 & 0 & 2.0000 & 0 & 0.1905 \\ 0 & 0 & 4.0000 & 7.0000 & 25.0000 & 4.0000 & 1.0000 & 0 & 0 & 2.0000 \\ 0 & 0 & 0 & 0 & 3.0000 & 14.0000 & 9.0000 & 0 & 0 & 0.1780 \\ 0 & 1.0000 & 4.0000 & 0.9524 & 2.0000 & 3.0000 & 12.0000 & 1.0000 & 1.0000 & 1.0287 \\ 1.0000 & 0.1000 & 5.0000 & 1.4905 & 1.6667 & 0 & 5.0000 & 10.0000 & 0.5260 & 0.9524 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.0000 & 0.6312 & 20.0000 & 0 \\ 0 & 2.0000 & 3.0000 & 0.7143 & 4.0000 & 0.5751 & 1.2801 & 0 & 0 & 35.0000 \end{bmatrix}$$



## Results:

Method	No pre-conditioning	Jacobi	SSOR	ILU(0)
iterations	10	7	4	4
error	3.1623	0.2338	0.1319	0.1300
	0.9439	0.0124	0.0190	0.0165
	0.2788	0.0032	0.0022	0.0012
	0.0948	0.0011	0.0002	0.0001
	0.0332	0.0003		
	0.0056	0.0001		
	0.0018	0.0000		
	0.0005			
	0.0003			
	0.0000			

Without preconditioning, the GMRES method takes a full 10 iterations to converge (n=10)

Fastest convergence, but takes the most computation to find  $M^{-1}$

# Why do I care?

1. Power Flow Project
  - A.  $Y_{bus}$ , Jacobian matrices – Newton Raphson
  - B. Factorization, Backward-Forward substitution for NR
  - C. Fast Decoupled
  - D. Q-limits, transformer taps
2. Sparse Power Flow Project
3. Continuation Power Flow Project
4. State Estimation Project
5. Optimal Power Flow Project

## 1. Power Flow Project (20%)

- A.  $Y_{bus}$ , Jacobian matrices – Newton Raphson
- B. LU Factorization, Backward-Forward substitution for NR
- C. Fast Decoupled
- D. Q-limits, transformer taps

- Using IEEE 14 bus system as given, and I will provide a slightly modified version after you confirm the 14-bus system.
- Projects build on each other, adding different features
- In addition to getting the right answers, for each program we'll set up a session for you to describe your code to me.
- Copying code from a colleague or from on-line resources is not permitted.

# Announcements

- Review examples from Chapter 2
- Read Sections 3.1-3.4
- Working on MATLAB Program
  - Input the CDF file – Start with Data in Program
  - Start working on building the Ybus
  - Calculate Jacobian Matrix Values
  - Work on LU Decomposition
- Extra Video will drop this week

## Student Hours

- Pullman this week
  - Tuesdays 4:30-5:30 pm (after class)
  - Wednesdays 4-5 pm
  - Fridays 1:30-2:30 pm
- Join from PC, Mac, Linux, iOS, or Android: <https://wsu.zoom.us/j/8237216735> (Links to an external site.)
- Meeting ID: 823 721 6735

Or by request via email [noel.Schulz@wsu.edu](mailto:noel.Schulz@wsu.edu) or phone 509-336-5522