Part 3
Chapter 12

Iterative Methods

School of Mechanical Engineering
Chung-Ang University

Chapter Objectives

- Understanding the difference between the Gauss-Seidel and Jacobi methods.
- Knowing how to assess diagonal dominance and knowing what it means.
- Recognizing how relaxation can be used to improve convergence of iterative methods.
- Understanding how to solve systems of nonlinear equations with successive substitution and Newton-Raphson.

Gauss-Seidel Method

- The Gauss-Seidel method is the most commonly used iterative method for solving linear algebraic equations [A]{x}={b}.
- For a 3x3 system with nonzero elements along the diagonal, for example, the j^{th} iteration values are found from the $j\text{-}1^{th}$ iteration using:

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

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

$$x_1^j = \frac{b_1 - a_{12}x_2^{j-1} - a_{13}x_3^{j-1}}{a_{11}}$$

$$x_2^j = \frac{b_2 - a_{21}x_1^{j} - a_{23}x_3^{j-1}}{a_{22}}$$

$$x_3^j = \frac{b_3 - a_{31}x_1^{j} - a_{32}x_2^{j}}{a_{33}}$$

Jacobi Iteration

- The Jacobi iteration is similar to the Gauss-Seidel method, except the j-1th information is used to update all variables in the jth iteration:

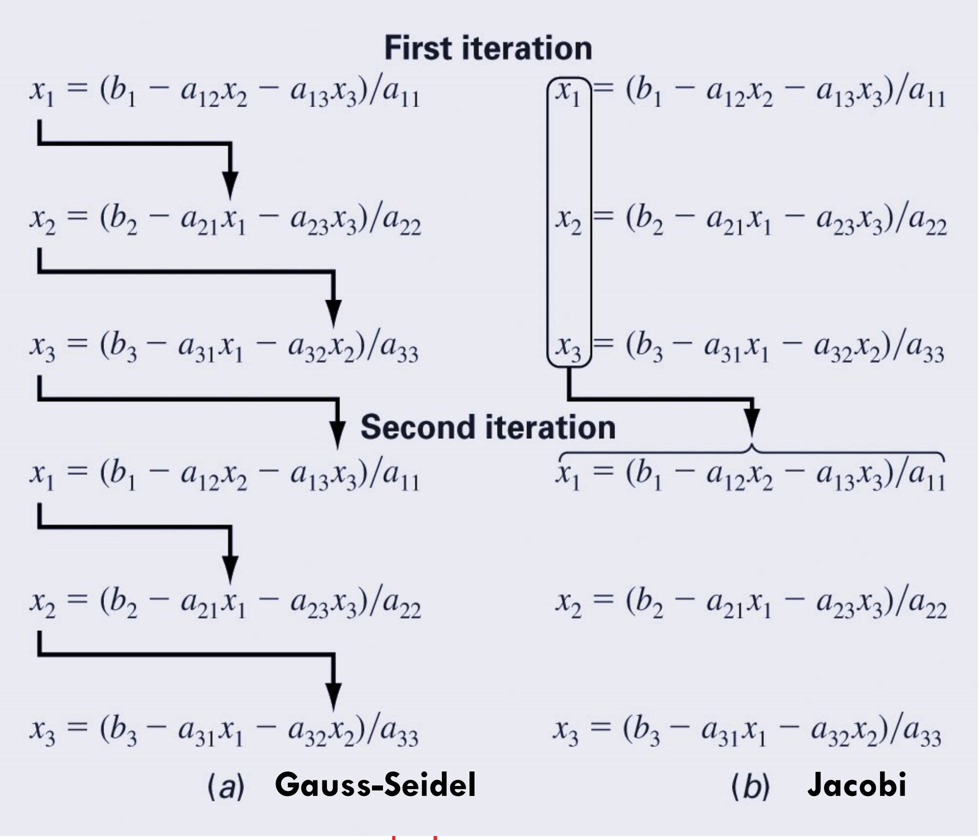
  a) Gauss-Seidel

  b) Jacobi

$$x_1^j = \frac{b_1 - a_{12}x_2^{j-1} - a_{13}x_3^{j-1}}{a_{11}}$$

$$x_2^j = \frac{b_2 - a_{21}x_1^{j-1} - a_{23}x_3^{j-1}}{a_{22}}$$

$$x_3^j = \frac{b_3 - a_{31}x_1^{j-1} - a_{32}x_2^{j-1}}{a_{33}}$$



Convergence

- The convergence of an iterative method can be calculated by determining the relative percent change of each element in {x}. For example, for the i^{th} element in the j^{th} iteration,

$$\varepsilon_{a,i} = \left| \frac{x_i^j - x_i^{j-1}}{x_i^j} \right| \times 100\%$$

- The method is ended when all elements have converged to a set tolerance.

Example 12.1

Q. Use the Gauss-Seidel Method to solve this set of equations.

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

Note : True solution is $\{x\}^T = \begin{bmatrix} 3 & -2.5 & 7 \end{bmatrix}$

Assume that x_2 and x_3 are zero in the first computation.

$$x_1 = \frac{7.85 + 0.1x_2 + 0.2x_3}{3} \qquad x_2 = \frac{-19.3 - 0.1x_1 + 0.3x_3}{7} \qquad x_3 = \frac{71.4 - 0.3x_1 + 0.2x_2}{10}$$

Example 12.1 (cont.)

$$x_1^{\,j} = \frac{7.85 + 0.1x_2^{\,j-1} + 0.2x_3^{\,j-1}}{3} \qquad x_2^{\,j} = \frac{-19.3 - 0.1x_1^{\,j} + 0.3x_3^{\,j-1}}{7}$$

$$x_3^{\,j} = \frac{71.4 - 0.3x_1^{\,j} + 0.2x_2^{\,j}}{10}$$

1st iteration

$$x_1 = \frac{7.85 + 0.1(0) + 0.2(0)}{3} = 2.616667$$

$$x_2 = \frac{-19.3 - 0.1(2.616667) + 0.3(0)}{7} = -2.794524$$

$$x_3 = \frac{71.4 - 0.3(2.616667) + 0.2(-2.794524)}{10} = 7.005610$$

Example 12.1 (cont.)

2nd iteration

$$x_1 = \frac{7.85 + 0.1(-2.794524) + 0.2(7.005610)}{3} = 2.990557$$

$$x_2 = \frac{-19.3 - 0.1(2.990557) + 0.3(7.005610)}{7} = -2.499625$$

$$x_3 = \frac{71.4 - 0.3(2.990557) + 0.2(-2.499625)}{10} = 7.000291$$

$$\varepsilon_{a,1} = \left| \frac{2.990557 - 2.616667}{2.990557} \right| \times 100\% = 12.5\%$$

$\varepsilon_{a,2} = 11.8\%$; $\varepsilon_{a,3} = 0.076\%$;

Diagonal Dominance

- The Gauss-Seidel method may diverge, but if the system is diagonally dominant, it will definitely converge.
- Diagonal dominance means:

$$\left|a_{ii}\right| > \sum_{\substack{j=1 \\ j \neq i}}^{n} \left|a_{ij}\right|$$

- Many engineering problems satisfy this requirement

MATLAB Program

MATLAB M-file: Gauss-Seidel

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

$$\{x\} = \{d\} - [C]\{x\}$$

$$\{d\} = \begin{Bmatrix} b_1 / a_{11} \\ b_2 / a_{22} \\ b_3 / a_{33} \end{Bmatrix} \qquad [C] = \begin{bmatrix} 0 & a_{12}/a_{11} & a_{13}/a_{11} \\ a_{21}/a_{22} & 0 & a_{23}/a_{22} \\ a_{31}/a_{33} & a_{32}/a_{33} & 0 \end{bmatrix}$$

```
function x = GaussSeidel(A,b,es,maxit)
% GaussSeidel: Gauss Seidel method
%   x = GaussSeidel(A,b): Gauss Seidel without relaxation
% input:
%   A = coefficient matrix
%   b = right hand side vector
%   es = stop criterion (default = 0.00001%)
%   maxit = max iterations (default = 50)
% output:
%   x = solution vector

if nargin<2,error('at least 2 input arguments required'),end
if nargin<4|isempty(maxit),maxit=50;end
if nargin<3|isempty(es),es=0.00001;end
[m,n] = size(A);
if m~=n, error('Matrix A must be square'); end
C = A;
for i = 1:n
  C(i,i) = 0;
  x(i) = 0;
end
x = x';
for i = 1:n
  C(i,1:n) = C(i,1:n)/A(i,i);
end
for i = 1:n
  d(i) = b(i)/A(i,i);
end
iter = 0;
while (1)
  xold = x;
  for i = 1:n
    x(i) = d(i)-C(i,:)*x;
    if x(i) ~= 0
      ea(i) = abs((x(i) - xold(i))/x(i)) * 100;
    end
  end
  iter = iter+1;
  if max(ea)<=es | iter >= maxit, break, end
end
```

Relaxation

- To enhance convergence, an iterative program can introduce relaxation where the value at a particular iteration is made up of a combination of the old value and the newly calculated value (update for the new one):

$$x_i^{\text{new}} = \lambda x_i^{\text{new}} + (1-\lambda) x_i^{\text{old}}$$

where λ is a weighting factor that is assigned a value between 0 and 2.

- 0< λ <1: underrelaxation
- λ =1: no relaxation
- 1< λ ≤2: overrelaxation

Nonlinear Systems

- Nonlinear systems can also be solved using the same strategy as the Gauss-Seidel method - solve each system for one of the unknowns and update each unknown using information from the previous iteration.

- This is called successive substitution.

Example 12.2

Q. Use successive substitution to determine the roots of the following equation. A correct pair of roots is $x_1 = 2$ and $x_2 = 3$.
Use the initial guesses of $x_1 = 1.5$ and $x_2 = 3.5$.

$$x_1^2 + x_1 x_2 = 10$$
$$x_2 + 3x_1 x_2^2 = 57$$

$$x_1 = \frac{10 - x_1^2}{x_2} \qquad x_2 = 57 - 3x_1 x_2^2$$

First iteration

$$x_1 = \frac{10 - (1.5)^2}{3.5} = 2.21429 \qquad x_2 = 57 - 3(2.21429)(3.5)^2 = -24.37516$$

Second iteration

$$x_1 = \frac{10 - (2.21429)^2}{-24.37516} = -0.20910 \qquad x_2 = 57 - 3(-0.20910)(-24.37516)^2 = 429.709$$

Seems to be diverging

Use the same equation but with a different format

$$x_1 = \sqrt{10 - x_1 x_2} \qquad x_2 = \sqrt{\frac{57 - x_2}{3x_1}}$$

First iteration

$$x_1 = \sqrt{10 - 1.5(3.5)} = 2.17945 \qquad x_2 = \sqrt{\frac{57 - 3.5}{3(2.17945)}} = 2.86051$$

Second iteration

$$x_1 = \sqrt{10 - 2.17945(2.86051)} = 1.94053$$

$$x_2 = \sqrt{\frac{57 - 2.86051}{3(1.94053)}} = 3.04955$$

The approach is converging on the true values.

→ The most serious shortcoming of substitution, which depends on the manner in which the equations are formulated.

Newton-Raphson

- Nonlinear systems may also be solved using the Newton-Raphson method for multiple variables.
- For a one-variable system, the Taylor series approximation and resulting Newton-Raphson equations are:

$$f(x_{i+1}) = f(x_i) + (x_{i+1} - x_i) f'(x_i) \qquad x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

- For a two-variable system,

$$f_{1,i+1} = f_{1,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{1,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{1,i}}{\partial x_2} \qquad x_{1,i+1} = x_{1,i} - \frac{f_{1,i}\frac{\partial f_{2,i}}{\partial x_2} - f_{2,i}\frac{\partial f_{1,i}}{\partial x_2}}{\frac{\partial f_{1,i}}{\partial x_1}\frac{\partial f_{2,i}}{\partial x_2} - \frac{\partial f_{1,i}}{\partial x_2}\frac{\partial f_{2,i}}{\partial x_1}}$$

$$f_{2,i+1} = f_{2,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{2,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{2,i}}{\partial x_2} \qquad x_{2,i+1} = x_{2,i} - \frac{f_{2,i}\frac{\partial f_{1,i}}{\partial x_1} - f_{1,i}\frac{\partial f_{2,i}}{\partial x_1}}{\frac{\partial f_{1,i}}{\partial x_1}\frac{\partial f_{2,i}}{\partial x_2} - \frac{\partial f_{1,i}}{\partial x_2}\frac{\partial f_{2,i}}{\partial x_1}}$$

$$f_{1,i+1} = f_{1,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{1,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{1,i}}{\partial x_2}$$

$$f_{2,i+1} = f_{2,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{2,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{2,i}}{\partial x_2}$$

$$[Z]\{x_{i+1}\} = -\{f\} + [Z]\{x_i\}, \quad \text{where } [Z] = Jacobian\ matrix$$

$$[Z]\{x_{i+1} - x_i\} = -\{f\}$$

$$[Z] = \begin{bmatrix} \frac{\partial f_{1,i}}{\partial x_1} & \frac{\partial f_{1,i}}{\partial x_2} & \cdots & \frac{\partial f_{1,i}}{\partial x_n} \\ \frac{\partial f_{2,i}}{\partial x_1} & \frac{\partial f_{2,i}}{\partial x_2} & \cdots & \frac{\partial f_{2,i}}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_{n,i}}{\partial x_1} & \frac{\partial f_{n,i}}{\partial x_2} & \cdots & \frac{\partial f_{n,i}}{\partial x_n} \end{bmatrix}$$

$$\{x_i\}^T = \left[x_{1,i} \quad x_{2,i} \quad \cdots \right.$$

$$\{x_{i+1}\}^T = \left[x_{1,i+1} \quad x_{2,i+1} \quad \cdots \quad \right]$$

$$\{f\}^T = \left[f_{1,i} \quad f_{2,i} \quad \cdots \right.$$

$$\text{Cramer's rule} \rightarrow \left(\begin{array}{l} x_{1,i+1} = x_{1,i} - \dfrac{f_{1,i} \dfrac{\partial f_{2,i}}{\partial x_2} - f_{2,i} \dfrac{\partial f_{1,i}}{\partial x_2}}{Jocobian} \\ x_{2,i+1} = x_{2,i} - \dfrac{f_{2,i} \dfrac{\partial f_{1,i}}{\partial x_1} - f_{1,i} \dfrac{\partial f_{2,i}}{\partial x_1}}{Jocobian} \end{array} \right)$$

$$Jocobian = \frac{\partial f_{1,i}}{\partial x_1} \frac{\partial f_{2,i}}{\partial x_2} - \frac{\partial f_{1,i}}{\partial x_2} \frac{\partial f_{2,i}}{\partial x_1}$$

Example 12.3 (1/2)

Q. Use the Newton-Raphson method to determine the roots of the equations.
Use the initial guesses of x1 =1.5 and x2 = 3.5.

$$x_1^2 + x_1 x_2 = 10$$

$$x_2 + 3x_1 x_2^2 = 57$$

$$\frac{\partial f_{1,0}}{\partial x_1} = 2x_1 + x_2 = 2(1.5) + 3.5 = 6.5 \qquad \frac{\partial f_{1,0}}{\partial x_2} = x_1 = 1.5$$

$$\frac{\partial f_{2,0}}{\partial x_1} = 3x_2^2 = 3(3.5)^2 = 36.75 \qquad \frac{\partial f_{2,0}}{\partial x_2} = 1 + 6x_1 x_2 = 1 + 6(1.5)(3.5) = 32.5$$

$$Jacobian = 6.5(32.5) - 1.5(36.75) = 156.125$$

Example 12.3 (2/2)

The values of the functions can be evaluated at the initial guesses as

$$f_{1,0} = (1.5)^2 + 1.5(3.5) - 10 = -2.5$$

$$f_{2,0} = 3.5 + 3(1.5)(3.5)^2 - 57 = 1.625$$

These values can be substituted to give

$$x_1 = 1.5 - \frac{-2.5(32.5) - 1.625(1.5)}{156.125} = 2.03603$$

$$x_2 = 3.5 - \frac{1.625(6.5) - (-2.5)(36.75)}{156.125} = 2.84388$$

The computation can be repeated until an acceptable accuracy is obtained.

MATLAB Program

```
function [x,f,ea,iter]=newtmult(func,x0,es,maxit,varargin)
% newtmult: Newton-Raphson root zeroes nonlinear systems
%  [x,f,ea,iter]=newtmult(func,x0,es,maxit,p1,p2,...):
%    uses the Newton-Raphson method to find the roots of
%    a system of nonlinear equations
% input:
%   func = name of function that returns f and J
%   x0 = initial guess
%   es = desired percent relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
%   p1,p2,... = additional parameters used by function
% output:
%   x = vector of roots
%   f = vector of functions evaluated at roots
%   ea = approximate percent relative error (%)
%   iter = number of iterations

if nargin<2,error('at least 2 input arguments required'),end
if nargin<3|isempty(es),es=0.0001;end
if nargin<4|isempty(maxit),maxit=50;end
iter = 0;
x=x0;
while (1)
  [J,f]=func(x,varargin{:});
  dx=J\f;
  x=x-dx;
  iter = iter + 1;
  ea=100*max(abs(dx./x));
  if iter>=maxit|ea<=es, break, end
end
```