Chapter 12

Iterative Method

Numerical Methods
Fall 2019

Gauss-Seidel Method

- The *Gauss-Seidel method* is the most commonly used iterative method for solving linear algebraic equations $[A]\{x\}=\{b\}$.
- The method solves each equation in a system for a particular variable, and then uses that value in later equations to solve later variables.
- For a 3×3 system with nonzero elements along the diagonal, for example, the j^{th} iteration values are found from the $j-1^{\text{th}}$ iteration using:

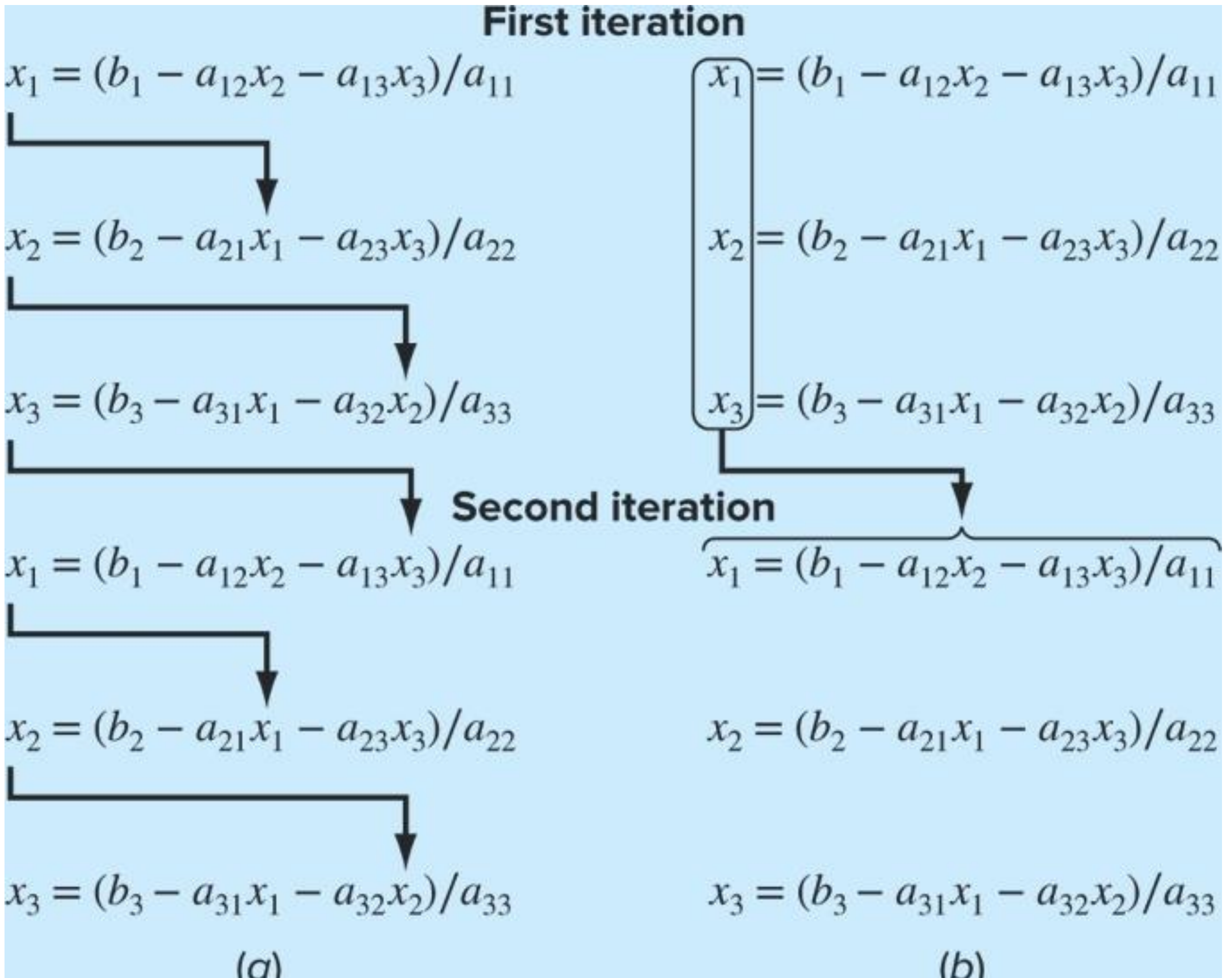
$$x_1^j = \frac{b_1 - a_{12}x_2^{j-1} - a_{13}x_3^{j-1}}{a_{11}}$$

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

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

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:



Gauss–Seidel

Jacobi

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.

Diagonal Dominance

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

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

- That is, the absolute value of the diagonal element is greater than the sum of the absolute values of the off-diagonal elements

Example 12.1

MATLAB Program, 1

```
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:

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

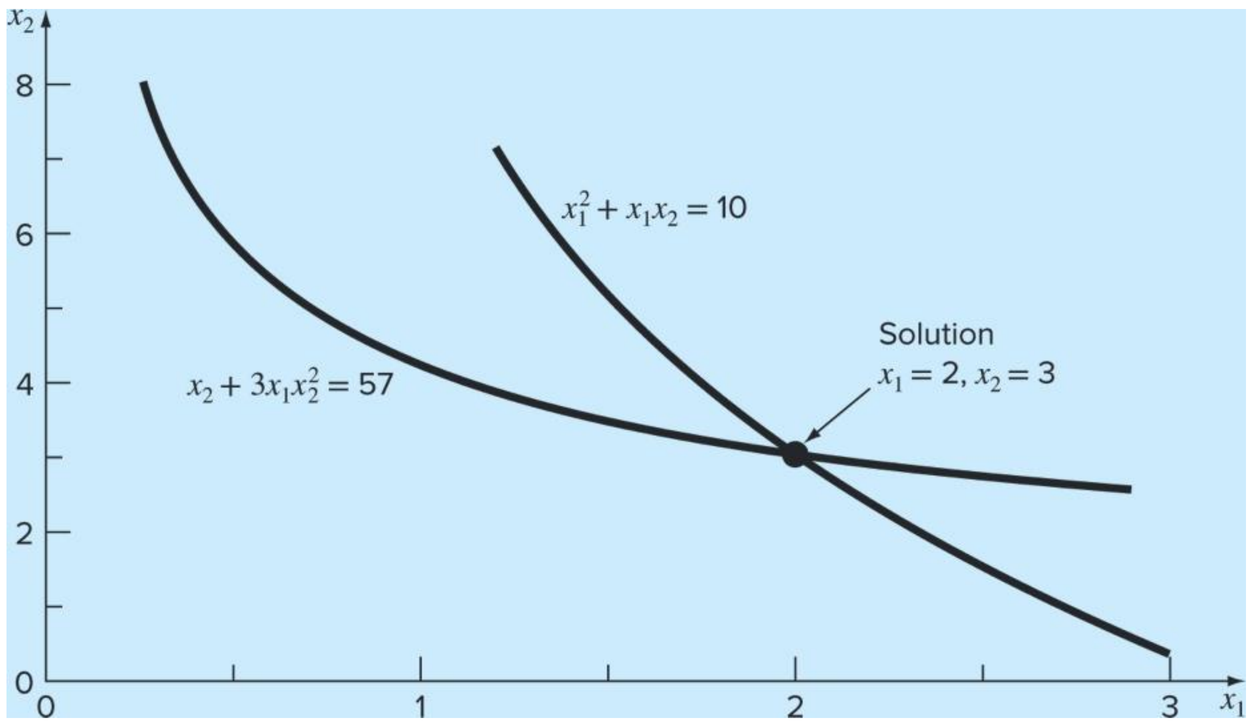
where λ is a weighting factor that is assigned a value between 0 and 2.

  - $0 < \lambda < 1$: underrelaxation → – Fast convergence – May fail to find true solution
  - $\lambda = 1$: no relaxation
  - $1 < \lambda \leq 2$: overrelaxation → – Slow convergence – Better to find true solution

Example 12.2

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 12.3

Newton-Raphson

- Nonlinear systems may also be solved using the Newton-Raphson method for multiple variables.
- A first-order Taylor series expansion:

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

where x_i is the initial guess at the root and x_{i+1} is the point at which the slope intercepts the x axis.

- At this intercept, $f(x_{i+1}) = 0$.

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

Newton-Raphson

- For a two-variable system, the Taylor series approximation and resulting Newton-Raphson equations are:

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

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

$$\Downarrow$$

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

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

Newton–Raphson

- Because all values subscripted with *i*'s are known (they correspond to the latest guess or approximation), the only unknowns are $x_{1,i+1}$ and $x_{2,i+1}$.

$$x_{1,i+1} = x_{1,i} - \frac{f_{1,i}\dfrac{\partial f_{2,i}}{\partial x_2} - f_{2,i}\dfrac{\partial f_{1,i}}{\partial x_2}}{\dfrac{\partial f_{1,i}}{\partial x_1}\dfrac{\partial f_{2,i}}{\partial x_2} - \dfrac{\partial f_{1,i}}{\partial x_2}\dfrac{\partial f_{2,i}}{\partial x_1}}$$

$$x_{2,i+1} = x_{2,i} - \frac{f_{2,i}\dfrac{\partial f_{1,i}}{\partial x_1} - f_{1,i}\dfrac{\partial f_{2,i}}{\partial x_1}}{\dfrac{\partial f_{1,i}}{\partial x_1}\dfrac{\partial f_{2,i}}{\partial x_2} - \dfrac{\partial f_{1,i}}{\partial x_2}\dfrac{\partial f_{2,i}}{\partial x_1}}$$

Example 12.4

MATLAB's `fsolve` Function

- The `fsolve` function solves systems of nonlinear equations with several variables. Its syntax is

```
[x, fx]  = fsolve(function, x0, options)
```

where
  - `[x, fx]` = a vector containing the roots `x` and a vector `fx` containing the values of the functions evaluated at the roots,
  - `function` = the name of the function containing a vector holding the equations being solved,
  - `x0` = a vector holding the initial guesses for the unknowns, and
  - `options` = a data structure created by the `optimset` function.
- Note that if you desire to pass function parameters but not use the `options`, pass an empty vector `[]` in its place.

fsolve Example

Solve:

$$f(x_1, x_2) = 2x_1 + x_1 x_2 - 10$$

$$f(x_1, x_2) = x_2 + 3x_1 x_2^2 - 57$$

Function to hold the equations:

```
function f = fun(x)
f = [x(1)^2+x(1)*x(2)-10;x(2)+3*x(1)*x(2)^2-57];
```

Script to generate the solution:

```
clc, format compact
[x,fx] = fsolve(@fun,[1.5;3.5])
```

Result:

```
x =
      2.0000
      3.0000
fx =
   1.0e-13 *
           0
      0.1421
```

Jacobian Matrix

- *Jacobian matrix* consisting of the partial derivatives of function f_1, $_{f2}$, f_3,… with variables x_1, x_2, x_3, …

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

Jacobian Matrix

- Solving the Newton-Raphson method using the Jacobian Matrix

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

Single function

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

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

Multi-function

$$\Rightarrow \quad \{x_{i+1}\} = \{x_i\} - [J]^{-1}\{f\}$$

Jacobian Matrix

- Example:

```
>> x=[1.5;3.5];
>> J=[2*x(1)+x(2) x(1);3*x(2)^2 1+6*x(1)*x(2)]

J =
    6.5000    1.5000
   36.7500   32.5000

>> f=[x(1)^2+x(1)*x(2)-10;x(2)+3*x(1)*x(2)^2-57]

f =
   -2.5000
    1.6250

>> x=x-J\f

x =
    2.0360
    2.8439
```