Systems of Linear Equations

Successive Over Relaxation method (iterative method)

Iterative Methods

- Iterative methods are more appropriate when
  - the <u>number of equations involved is large</u> (typically of the order of 100 or more)

    and/or

  - the <u>matrix is sparse</u> (less memory requirements)

Jacobi Iteration

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \quad \Rightarrow \quad \begin{aligned} x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11} \\ x_2 &= (b_2 - a_{21}x_1 - a_{23}x_3)/a_{22} \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \end{aligned}$$

Step 1: Construct updating formula as

$$\boxed{\begin{aligned} x_1^{(i+1)} &= (b_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)})/a_{11} \\ x_2^{(i+1)} &= (b_2 - a_{21}x_1^{(i)} - a_{23}x_3^{(i)})/a_{22} \\ x_3^{(i+1)} &= (b_3 - a_{31}x_1^{(i)} - a_{32}x_2^{(i)})/a_{33} \end{aligned}}$$

Step 2: Starts with initial guesses $x_1^{(0)}, x_2^{(0)}, x_3^{(0)}$

Step 3: Iteratively compute $x_1^{(i+1)}, x_2^{(i+1)}, x_3^{(i+1)}$ based on the updating formula for $i = 0, 1, \ldots,$ until the x's converge

Gauss-Seidel Iteration

- Improved version of Jacobi Iteration

- Observation: If x's converge, then after computing

$$x_1^{(i+1)} = (b_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)})/a_{11}$$

$x_1^{(i+1)}$ should be a better approximation of x_1 than $x_1^{(i)}$.

Thus the updating formula of Gauss-Seidel Iteration becomes

$$\boxed{x_1^{(i+1)}} = (b_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)})/a_{11}$$

$$\boxed{x_2^{(i+1)}} = (b_2 - a_{21}\boxed{x_1^{(i+1)}} - a_{23}x_3^{(i)})/a_{22}$$

$$x_3^{(i+1)} = (b_3 - a_{31}\boxed{x_1^{(i+1)}} - a_{32}\boxed{x_2^{(i+1)}})/a_{33}$$

Stopping Criteria

- When the estimated percentage relative error of every x's is less then the acceptable error.

$$\left|\varepsilon_{a,i}\right| = \left|\frac{x_i^{(j)} - x_i^{(j-1)}}{x_i^{(j)}}\right| 100\% < \varepsilon_s, \quad \text{for all } i$$

Example: Gauss-Seidel Method

- Solve

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

$$(Answer: x_1 = 3, \quad x_2 = -2.5, \quad x_3 = 7)$$

- Use Gauss-Seidel Method and obtain the following updating equation.

$$x_1 = (7.85 + 0.1x_2 + 0.2x_3)/3$$

$$x_2 = (-19.3 - 0.1x_1 + 0.3x_3)/7$$

$$x_3 = (71.4 - 0.3x_1 + 0.2x_2)/10$$

Initially, $x_2 = x_3 = 0$

Example: Gauss-Seidel Method

x_1	x_2	x_3	
$\boxed{2.616667}$	0	0	
2.616667	$\boxed{-2.794524}$	0	
2.616667	2.794524	$\boxed{7.005610}$	
$\boxed{2.990557}$	2.794524	7.005610	$\lvert\varepsilon_t\rvert$ of $x_1 = 0.31\%$
2.990557	$\boxed{-2.499625}$	7.005610	$\lvert\varepsilon_t\rvert$ of $x_2 = 0.015\%$
2.990557	-2.499625	$\boxed{7.00291}$	$\lvert\varepsilon_t\rvert$ of $x_3 = 0.0042\%$

- What would happen if we swap the rows?

Convergence

- A simple way to determine the convergence is to inspect the diagonal elements.

- **All of the diagonal elements must be non-zero.**

- Convergence is guaranteed if the system is diagonally dominant.

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

- What if the matrix is not diagonally dominant?

Convergence

- Good initial guess can greatly improve the chance of convergence.
- Convergent rate depends on the properties of the matrix.
- The higher the condition number, the slower the convergent rate.

Exercise

How would you solve the following system of equations using the Gauss-Seidel method with guarantee of convergence?

$$\begin{aligned} 10x_1 + 2x_2 + 14x_3 &= 1 \\ -24x_1 + 2x_2 + 14x_3 &= 1 \\ 6x_1 - 30x_2 + 13x_3 &= 1 \end{aligned}$$

Solution

To guarantee convergence, the matrix on the L.H.S. of the equations must be diagonal dominant.

For the given system of equations, we can reorder the equations so that the system becomes

$$\begin{array}{cccc} -24x_1 & +2x_2 & +14x_3 & =1 \\ 6x_1 & -30x_2 & +13x_3 & =1 \\ 10x_1 & 2x_2 & +14x_3 & =1 \end{array}$$

and construct the updating formula as

$$\begin{aligned} x_1^{(i+1)} &= (1-2x_2^{(i)}-14x_3^{(i)})/-24 \\ x_2^{(i+1)} &= (1-6x_1^{(i+1)}-13x_3^{(i)})/-30 \\ x_3^{(i+1)} &= (1-10x_1^{(i+1)}-2x_2^{(i+1)})/14 \end{aligned}$$

Pseudocode for Gauss-Seidel Iteration

```
// Assume arrays start with index 1 instead of 0.
// a: Matrix A
// b: Vector b
// n: Dimension of the system of equations
// x: Vector x; contains initial values and
//    will be used to store the solution
// imax: maximum number of iterations allowed
// es: Acceptable percentage relative error
// lambda: Use to "relax" x's
Gauss_Seidel(a, b, n, x, imax, es, lambda) {

    // Normalize elements in every row by
    // dividing them by the diagonal
    // element. Doing so can save some
    // division operations later.
    for i = 1 to n {
        dummy = a[i, i]
        for j = 1 to n
            a[i,j] = a[i,j] / dummy
        b[i] = b[i] / dummy
    }
```

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

Pseudocode for Gauss-Seidel Iteration

```
// Assuming the initial guess (the values in x)
// is not reliable.
// So carry out one iteration of Gauss-Seidel
// without relaxation (to get a better approximation
// of x's)
for i = 1 to n {
    sum = b[i]
    for j = 1 to n {
        if (i != j)
            sum = sum - a[i,j] * x[j]
    }
    x[i] = sum
}

iter = 1;
```

```
    do {
        sentinel = 1 // Assuming iteration converges
                     // set to 0 if error is still big
        for i = 1 to n {
            old = x[i]  // Save the previous x

            // Compute new x and apply relaxation
            sum = b[i]
            for j = 1 to n
                if (i != j)
                    sum = sum - a[i,j] * x[j]

            x[i] = lambda * sum + (1 - lambda) * old

            // If "necessary", check if error is too big
            if (sentinel == 1 AND x[i] != 0) {
                ea = abs((x[i] - old) / x[i]) * 100
                if (ea > es)
                    sentinel = 0
            }
        }
        iter = iter + 1;
    } while (sentinel == 0 AND iter < imax)
}
```

Successive Over Relaxation (SOR)

- Introduce an additional parameter, ***ω***, that may accelerate the convergence of the iterations

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\
a_{21}x_1 + a_{22}x_1 + a_{23}x_3 &= b_2 \\
a_{31}x_1 + a_{32}x_1 + a_{33}x_3 &= b_3
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
\omega a_{11}x_1 + \omega a_{12}x_2 + \omega a_{13}x_3 &= \omega b_1 \\
\omega a_{21}x_1 + \omega a_{22}x_1 + \omega a_{23}x_3 &= \omega b_2 \\
\omega a_{31}x_1 + \omega a_{32}x_1 + \omega a_{33}x_3 &= \omega b_3
\end{aligned}
$$

$$x_1^{\,new} = (1-\omega)x_1^{\,old} + \frac{\omega}{a_{11}}(b_1 - a_{12}x_2^{\,old} - a_{13}x_3^{\,old})$$

$$x_2^{\,new} = (1-\omega)x_2^{\,old} + \frac{\omega}{a_{22}}(b_2 - a_{21}x_1^{\,new} - a_{23}x_3^{\,old})$$

$$x_3^{\,new} = (1-\omega)x_3^{\,old} + \frac{\omega}{a_{33}}(b_3 - a_{31}x_1^{\,new} - a_{32}x_2^{\,new})$$

Successive Over-Relaxation (SOR)

Successive Over-Relaxation is a modification of the Gauss-Seidel method to enhance convergence.

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

where

λ is a weighting factor between 0 and 2 (or use ω).

x_i is the value computed from the original Gauss-Seidel updating equation (i.e. before applying SOR).

x_i^{old} is the value of x_i in the previous iteration

x_i^{new} is the new value of x_i, used for updating.

Relaxation

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

- $\lambda = 1$, $x_i^{new} = x_i$
- underrelaxation, $0 \le \lambda < 1$

  to make a non-convergent system converge, or to speedup convergence by avoiding oscillations.

- overrelaxation, $1 < \lambda \le 2$

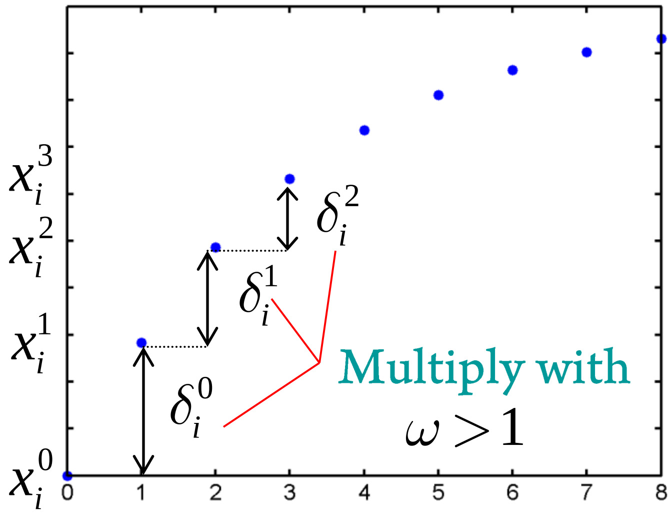
  to accelerate the convergence, if x_i is moving toward the solution at a slow rate. More weighing is given to the current value of x.

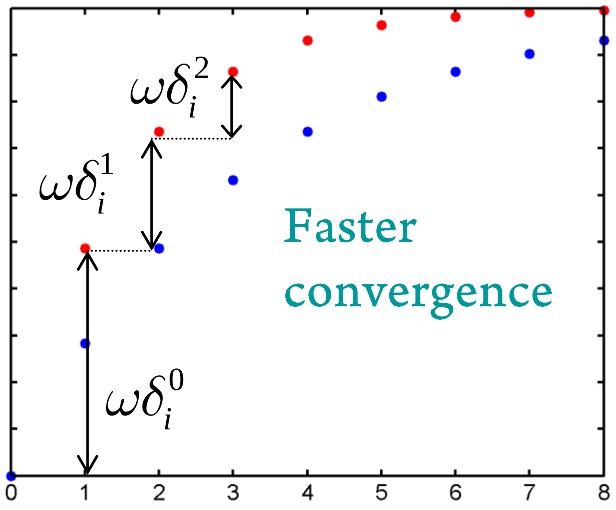
Successive Over Relaxation Method

- GS iteration can be also written as follows

$$x_i^{k+1} = x_i^k + \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i}^{n} a_{ij}x_j^k\right]$$

$$x_i^{k+1} = x_i^k + \boxed{\delta_i^k} \longrightarrow \text{Correction term}$$





SOR

$$x_i^{k+1} = x_i^k + \omega \delta_i^k$$

$$x_i^{k+1} = x_i^k + \omega \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i}^{n} a_{ij} x_j^k \right]$$

$$x_i^{k+1} = (1-\omega) x_i^k + \omega \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right]$$

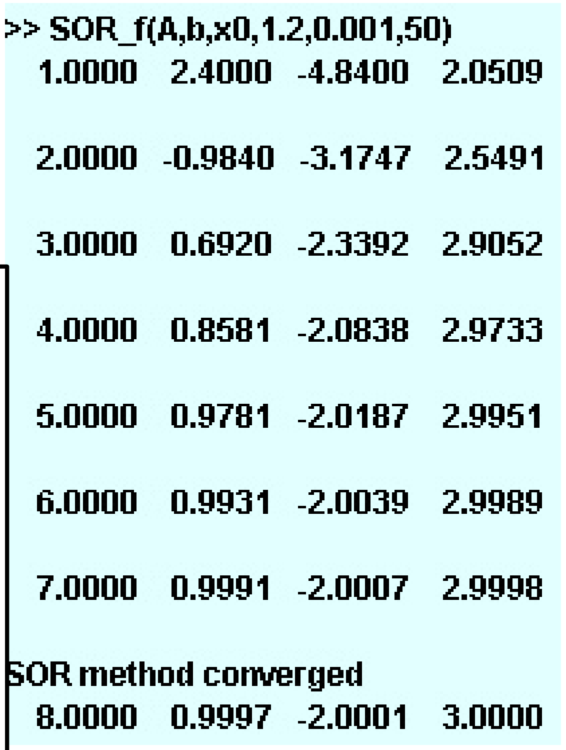
$1 < \boldsymbol{\omega} < 2$ over relaxation (faster convergence)
$0 < \boldsymbol{\omega} < 1$ under relaxation (slower convergence)
There is an optimum value for $\boldsymbol{\omega}$
Find it by trial and error (usually around 1.6)

Successive Over Relaxation (SOR)

- Consider the three-by-three system $Ax = b$

$$A = \begin{bmatrix} 4 & -2 & 0 \\ -2 & 6 & -5 \\ 0 & -5 & 11 \end{bmatrix} \qquad b = \begin{bmatrix} 8 \\ -29 \\ 43 \end{bmatrix}$$

$$\boxed{\begin{aligned} x_1^{new} &= (1-\omega)x_1^{old} + \omega\left(\frac{1}{2}x_2^{old} + 2\right) \\ x_2^{new} &= (1-\omega)x_2^{old} + \omega\left(\frac{1}{3}x_1^{new} + \frac{5}{6}x_3^{old} - \frac{29}{6}\right) \\ x_3^{new} &= (1-\omega)x_3^{old} + \omega\left(\frac{5}{11}x_1^{new} + \frac{43}{11}\right) \end{aligned}}$$



```
>> SOR_f(A,b,x0,1.2,0.001,50)
   1.0000   2.4000  -4.8400   2.0509

   2.0000  -0.9840  -3.1747   2.5491

   3.0000   0.6920  -2.3392   2.9052

   4.0000   0.8581  -2.0838   2.9733

   5.0000   0.9781  -2.0187   2.9951

   6.0000   0.9931  -2.0039   2.9989

   7.0000   0.9991  -2.0007   2.9998

SOR method converged
   8.0000   0.9997  -2.0001   3.0000
```

Successive Over Relaxation (SOR)

- Required number of iterations for different values of the relaxation parameter
  - Start with $x^{(0)} = [0\ \ 0\ \ \ldots 0]'$
  - Tolerance = 0.00001

ω	0.8	0.9	1.0	1.2	1.25	1.3	1.4
No. of iterations	44	36	29	18	15	13	16

Example of SOR

$$4X_1 + 2X_2 = 2$$

$$2X_1 + 10X_2 + 4X_3 = 6$$

$$4X_2 + 5X_3 = 5$$

Solution: $(X_1, X_2, X_3) = (0.41379, 0.17241, 0.86206)$

SOR Example

- Formulation of the SOR Algorithm

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}_{\text{new}} = (1-\omega)\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}_{\text{old}} + \omega\left[\begin{Bmatrix} 0.5 \\ 0.6 \\ 1.0 \end{Bmatrix} - \begin{bmatrix} 0 & 0.5 & 0 \\ 0.2 & 0 & 0.4 \\ 0 & 0.8 & 0 \end{bmatrix}\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}_{\text{combined}}\right]$$

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}_{\text{combined}} = \begin{Bmatrix} \{x\}_{\text{new}} \\ \{x\}_{\text{old}} \end{Bmatrix}$$

Effects of w Parameter

- Using **nmax=50** and **tol = 0.000001**

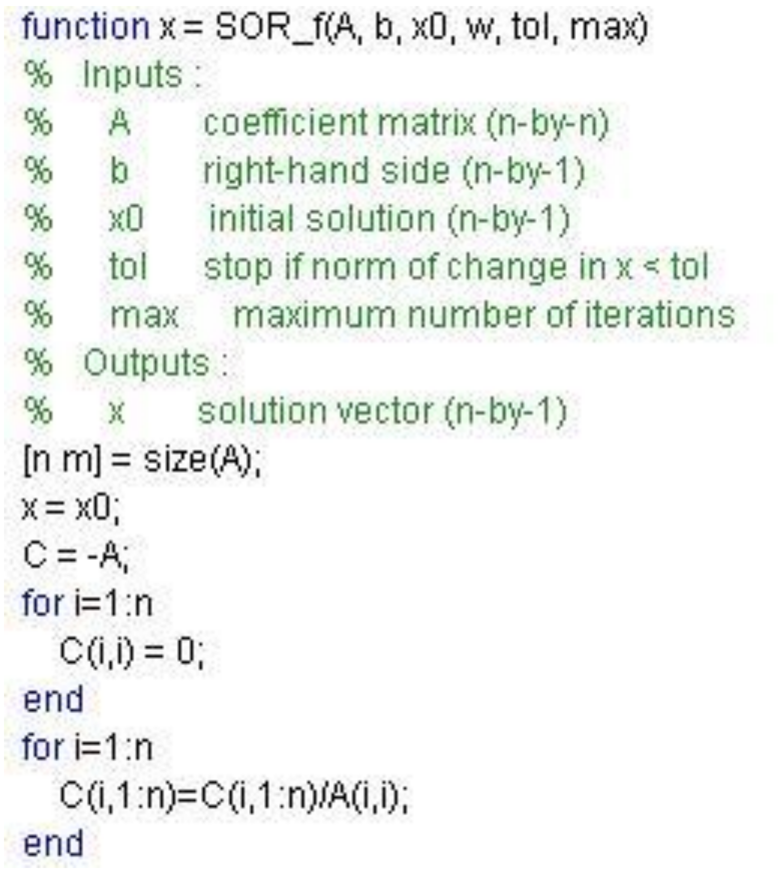
W	Number of Iterations		W	Number of Iterations
0,7	33		1,25	12
0,8	27		1,3	14
0,9	22		1,4	17
1	17		1,5	22
1,1	13		1,6	30
1,15	10		1,7	43
1,175	10			
1,2	10			

How to determine λ?

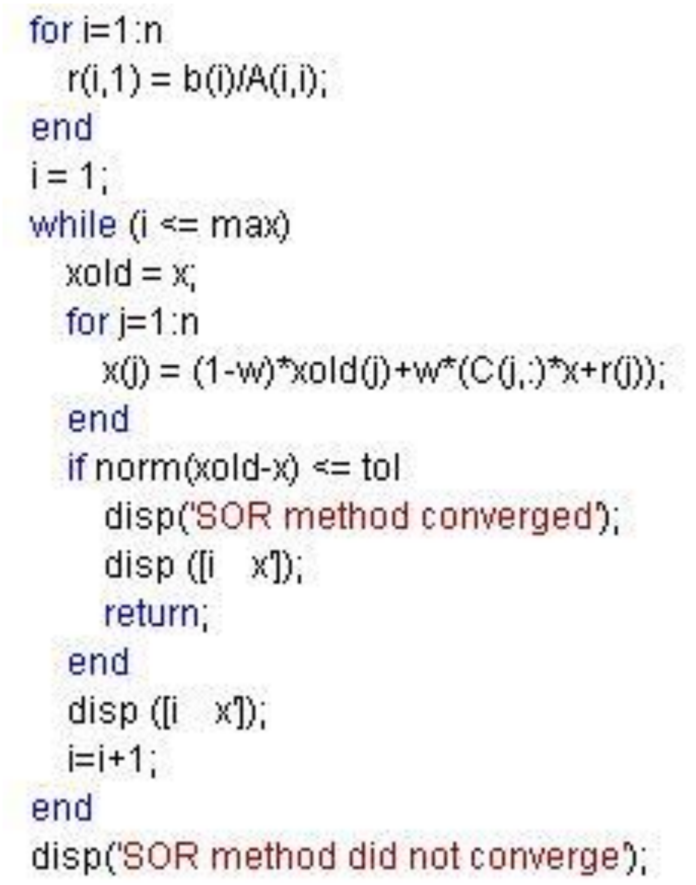
$$x_i^{new} = \lambda x_i + (1-\lambda)x_i^{old}$$

- Problem specific
- Usually determined empirically
- Not useful when the system is only solved once
- If the system are solved many times, a carefully selected value of λ can greatly improve the rate of convergence.

Matlab function for SOR



```
function x = SOR_f(A, b, x0, w, tol, max)
%  Inputs :
%     A       coefficient matrix (n-by-n)
%     b       right-hand side (n-by-1)
%     x0      initial solution (n-by-1)
%     tol     stop if norm of change in x < tol
%     max     maximum number of iterations
%  Outputs :
%     x       solution vector (n-by-1)
[n m] = size(A);
x = x0;
C = -A;
for i=1:n
   C(i,i) = 0;
end
for i=1:n
   C(i,1:n)=C(i,1:n)/A(i,i);
end
```



```
for i=1:n
   r(i,1) = b(i)/A(i,i);
end
i = 1;
while (i <= max)
   xold = x;
   for j=1:n
      x(j) = (1-w)*xold(j)+w*(C(j,:)*x+r(j));
   end
   if norm(xold-x) <= tol
      disp('SOR method converged');
      disp ([i    x']);
      return;
   end
   disp ([i    x']);
   i=i+1;
end
disp('SOR method did not converge');
```

Summary (3 X 3 system)

- Jacobi method

$$
\begin{aligned}
x_1^{(k)} &= -\frac{a_{12}}{a_{11}} x_2^{(k-1)} - \frac{a_{13}}{a_{11}} x_3^{(k-1)} + \frac{b_1}{a_{11}} \\
x_2^{(k)} &= -\frac{a_{21}}{a_{22}} x_1^{(k-1)} - \frac{a_{23}}{a_{22}} x_3^{(k-1)} + \frac{b_2}{a_{22}} \\
x_3^{(k)} &= -\frac{a_{31}}{a_{33}} x_1^{(k-1)} - \frac{a_{32}}{a_{33}} x_2^{(k-1)} + \frac{b_3}{a_{33}}
\end{aligned}
$$

- Gauss-seidel method

$$
\begin{aligned}
x_1^{(k)} &= -\frac{a_{12}}{a_{11}} x_2^{(k-1)} - \frac{a_{13}}{a_{11}} x_3^{(k-1)} + \frac{b_1}{a_{11}} \\
x_2^{(k)} &= -\frac{a_{21}}{a_{22}} x_1^{(k)} - \frac{a_{23}}{a_{22}} x_3^{(k-1)} + \frac{b_2}{a_{22}} \\
x_3^{(k)} &= -\frac{a_{31}}{a_{33}} x_1^{(k)} - \frac{a_{32}}{a_{33}} x_2^{(k-1)} + \frac{b_3}{a_{33}}
\end{aligned}
$$

- SOR method

$$
\begin{aligned}
x_1^{new} &= (1-\omega)x_1^{old} + \frac{\omega}{a_{11}}(b_1 - a_{12}x_2^{old} - a_{13}x_3^{old}) \\
x_2^{new} &= (1-\omega)x_2^{old} + \frac{\omega}{a_{22}}(b_2 - a_{21}x_1^{new} - a_{23}x_3^{old}) \\
x_3^{new} &= (1-\omega)x_3^{old} + \frac{\omega}{a_{33}}(b_3 - a_{31}x_1^{new} - a_{32}x_2^{new})
\end{aligned}
$$

Summary

Method	Algorithm for performing iteration $k+1$: For $i = 1$ to n do:
Jacobi	$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}\right)$
Gauss-Seidel	$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}\right)$
SOR	$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}\right)$

Exercise

Use Gauss-Seidel method

- (a) without relaxation and
- (b) with relaxation $(\lambda = 0.95)$

to solve the following system to a tolerance of $\varepsilon = 5\%$

If necessary, rearrange the equations to achieve convergence.

$$-3x_1 + x_2 + 12x_3 = 50$$
$$6x_1 - x_2 - x_3 = 3$$
$$6x_1 + 9x_2 + x_3 = 40$$

The system can be re-arranged to the following

$$6x_1 - x_2 - x_3 = 3$$
$$6x_1 + 9x_2 + x_3 = 40$$
$$-3x_1 + x_2 + 12x_3 = 50$$

Gauss-Seidel

$$x_1 = \frac{3 + x_2 + x_3}{6}$$

$$x_2 = \frac{40 - 6x_1 - x_3}{9}$$

$$x_3 = \frac{50 + 3x_1 - x_2}{12}$$

Without Relaxation

x1	x2	x3	ea1	ea2	ea3	maximum ea%
0	0	0				
0.5	4.111111	3.949074				
1.843364	2.776749	4.396112	0.728757	0.480548	0.101689	72.88%
1.695477	2.82567	4.355063	0.087225	0.017313	0.009425	8.72%
1.696789	2.829356	4.355084	0.000773	0.001303	4.78E-06	0.13%

with relaxation		lamda	0.95						
x1	x2	x3	x1'	x2'	x3'	ea1	ea2	ea3	max
0	0	0	0	0	0				
0.5	4.127778	3.958634	0.475	3.921389	3.760703				
1.813337	2.861475	4.359852	1.74767	2.92479	4.339791	0.72821	0.340742	0.133437	72.82%
1.714107	2.8162	4.361562	1.719068	2.818464	4.361476	0.016638	0.037725	0.004972	3.77%