Classical Iterative Methods for the Solution of Linear Systems

<u>Sugestão de referência</u>

- **David S. Watkins, 3rd, Chapter 8**
- **Lloyd N. Trefethen, David Bau III, Lecture 38. Conjugate Gradients**
- **Yousef Saad, 2nd ed, Chapter 4**

Classical Iterative Methods for the Solution of Linear Systems

Virtually all methods for solving $A\boldsymbol{x} = \boldsymbol{b}$ or $A\boldsymbol{x} = \lambda\boldsymbol{x}$ require $\mathcal{O}(m^3)$ operations. In practical applications A often has a certain structure and/or is *sparse*, i.e., A contains many zeros.

A typical problem that arises in practice is the Poisson problem mentioned at the beginning of the class. We want to find u such that

$$\begin{aligned} -\nabla^2 u(x,y) = -\left[u_{xx}(x,y) + u_{yy}(x,y)\right] = f(x,y), \qquad & \text{in } \Omega = [0,1]^2 \\ u(x,y) = 0, \qquad & \text{on } \partial\Omega. \end{aligned}$$

One of the standard numerical algorithms is a finite difference approach. The Laplacian is discretized on a grid of $(n+1)^2$ equally spaced points $(x_i, y_j) = (ih, jh)$, $i, j = 0, \ldots, n$ with $h = \frac{1}{n}$. This results in the discrete Laplacian

$$\nabla^2 u(x_i, y_j) \approx \frac{u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1} - 4u_{i,j}}{h^2},$$

where $u_{i,j} = u(x_i, y_j)$.

Classical Iterative Methods for the Solution of Linear Systems

The boundary conditions of the PDE allow us to set the solution at the points along the boundary as

$$u_{i,0} = u_{i,n} = u_{0,j} = u_{n,j} = 0, \qquad i,j = 0,1,\ldots,n.$$

At the $(n-1)^2$ interior grid points we obtain the following system of linear equations for the values of u there

$$4u_{i,j} - u_{i-1,j} - u_{i,j-1} - u_{i+1,j} - u_{i,j+1} = \frac{f_{i,j}}{n^2}, \qquad i,j = 1,\ldots,n-1$$

The system matrix is of size $m \times m$, where $m = (n-1)^2$. Each row contains at most five nonzero entries, and therefore is very sparse. Thus, special methods are called for to take advantage of this sparsity when we solve this linear system. Obviously, a full-blown LU or Cholesky factorization will be much too costly if m is large (typical values for m are often 10^6 or even larger).

The Splitting Approach

The basic iterative scheme to solve $A\boldsymbol{x} = \boldsymbol{b}$ will be of the form

$$\boldsymbol{x}^{(k)} = G\boldsymbol{x}^{(k-1)} + \boldsymbol{c}, \qquad k = 1, 2, 3, \ldots. \tag{1}$$

Here we assume that $A \in \mathbb{C}^{m\times m}$, $\boldsymbol{x}^{(0)}$ is an initial guess for the solution, and G and $\boldsymbol{c}$ are a constant iteration matrix and vector, respectively, defining the iterative scheme.

Most classical iterative methods are based on a *splitting* of the matrix A of the form

$$A = M - N$$

with a nonsingular matrix M. One then defines

$$G = M^{-1}N \quad \text{and} \quad \boldsymbol{c} = M^{-1}\boldsymbol{b}.$$

Then (38) becomes

$$\boldsymbol{x}^{(k)} = M^{-1}N\boldsymbol{x}^{(k-1)} + M^{-1}\boldsymbol{b}$$

or

$$M\boldsymbol{x}^{(k)} = N\boldsymbol{x}^{(k-1)} + \boldsymbol{b}. \tag{2}$$

In practice we will want to choose the splitting factors so that

The Splitting Approach

$$\boldsymbol{x}^{(k)} = M^{-1}N\boldsymbol{x}^{(k-1)} + M^{-1}\boldsymbol{b}$$

$$(2)$$

$$M\boldsymbol{x}^{(k)} = N\boldsymbol{x}^{(k-1)} + \boldsymbol{b}.$$

In practice we will want to choose the splitting factors so tha

1. (2) is easily solved,
2. (2) converges rapidly.

The Splitting Approach

Theorem *If*

$$\|G\| = \|M^{-1}N\| < 1$$

then (1) *converges to a solution of* $A\boldsymbol{x} = \boldsymbol{b}$ *for any initial guess* $\boldsymbol{x}^{(0)}$.

Proof (1) describes a fixed point iteration (i.e., is of the form $x = g(x)$), and the fixed point of (1) is a solution of $A\boldsymbol{x} = \boldsymbol{b}$ as can be seen from

$$\begin{aligned} & \boldsymbol{x} = G\boldsymbol{x} + \boldsymbol{c} \\ \iff \quad & \boldsymbol{x} = M^{-1}N\boldsymbol{x} + M^{-1}\boldsymbol{b} \\ \iff \quad & M\boldsymbol{x} = N\boldsymbol{x} + \boldsymbol{b} \\ \iff \quad & \underbrace{(M - N)}_{=A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned}$$

The Splitting Approach

Now we let $\boldsymbol{e}^{(k)} = \boldsymbol{x}^{(k)} - \boldsymbol{x}$, where $\boldsymbol{x}$ is the solution of the fixed point problem, and show that this quantity goes to zero as $k \to \infty$. First we observe that

$$\begin{aligned} \boldsymbol{e}^{(k)} &= \boldsymbol{x}^{(k)} - \boldsymbol{x} \\ &= G\boldsymbol{x}^{(k-1)} - G\boldsymbol{x} \\ &= G\left(\boldsymbol{x}^{(k-1)} - \boldsymbol{x}\right) \\ &= G\boldsymbol{e}^{(k-1)}. \end{aligned}$$

Taking norms we have

$$\begin{aligned} \|\boldsymbol{e}^{(k)}\| &= \|G\boldsymbol{e}^{(k-1)}\| \\ &\leq \|G\|\|\boldsymbol{e}^{(k-1)}\| \\ &\leq \|G\|^k\|\boldsymbol{e}^{(0)}\|, \end{aligned}$$

where the last inequality is obtained by recursion.

The Splitting Approach

If now – as we assume – $\|G\| < 1$, then $\|\boldsymbol{e}^{(k)}\| \to 0$ as $k \to \infty$, and therefore $\boldsymbol{x}^{(k)} \to \boldsymbol{x}$ and the method converges. ■

With some more effort one can show

Theorem *The iteration* (1) *converges to the solution of* $A\boldsymbol{x} = \boldsymbol{b}$ *for any initial guess* $\boldsymbol{x}^{(0)}$ *if and only if* $\rho(G) = \rho(M^{-1}N) < 1$.

Here $\rho(G)$ is the *spectral radius* of G, i.e., the largest eigenvalue of G (in modulus).

How should we choose M and N ?

We formally decompose $A = L + D + U$ into a lower triangular, diagonal, and upper triangular part. Then we let

$$M = D, \qquad N = -(L+U).$$

Thus (2) becomes

$$D\boldsymbol{x}^{(k)} = -(L+U)\boldsymbol{x}^{(k-1)} + \boldsymbol{b}$$

or

$$\boldsymbol{x}^{(k)} = D^{-1}\left[\boldsymbol{b} - (L+U)\boldsymbol{x}^{(k-1)}\right].$$

By writing this formula componentwise we have

$$x_i^{(k)} = \frac{b_i - \sum_{j\neq i} a_{ij} x_j^{(k-1)}}{a_{ii}}, \qquad i = 1, \ldots, m.$$

How should we choose M and N ?

This means we have the following algorithm.

Algorithm (Jacobi method)

```
Let $\boldsymbol{x}^{(0)}$ be an arbitrary initial guess
for $k = 1, 2, \ldots$
    for $i = 1 : m$
        $x_i^{(k)} = \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{m} a_{ij} x_j^{(k-1)} \right) / a_{ii}$
    end
end
```

How should we choose M and N ?

Example If we apply the Jacobi method to the finite difference discretization of the Poisson problem then we can be more efficient by taking advantage of the matrix structure. The central part of the algorithm (the loop for $i = 1 : m$) can then be replaced by

```
for $i = 1 : n - 1$
    for $j = 1 : n - 1$
        $u_{i,j}^{(k)} = \left( u_{i-1,j}^{(k-1)} + u_{i,j-1}^{(k-1)} + u_{i+1,j}^{(k-1)} + u_{i,j+1}^{(k-1)} + \frac{f_{i,j}}{n^2} \right) /4$
    end
end
```

Note that the unknowns are now u_{ij} instead of x_i. This algorithm can be implemented in one line of Matlab

How should we choose M and N ? *The Jacobi method*

Remark While the Jacobi method is not used that often in practice on serial computers it does lend itself to a naturally parallel implementation.

In order to get a convergence result for the Jacobi method we need to recall the concept of *diagonal dominance*. We say a matrix A is *strictly row diagonally dominant* if

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$

The Jacobi method

Theorem *If A is strictly row diagonally dominant, then the Jacobi method converges for any initial guess $\boldsymbol{x}^{(0)}$.*

Proof By the definition of diagonal dominance above we have

$$|a_{ii}| > \sum_{j\neq i} |a_{ij}| \quad \iff \quad \sum_{j\neq i} \frac{|a_{ij}|}{|a_{ii}|} < 1.$$

We are done if we can show that $\|G\| < 1$. Here

$$\|G\| = \|M^{-1}N\| = \|D^{-1}(L+U)\|.$$

Since we can take any norm, we pick $\|\cdot\|_\infty$. Then

$$\begin{aligned} \|G\|_\infty &= \|D^{-1}(L+U)\|_\infty \\ &= \max_{1\leq i\leq m} \sum_{j\neq i} \frac{|a_{ij}|}{|a_{ii}|} \\ &< 1 \end{aligned}$$

by the diagonal dominance. ■

The Jacobi method

Remark An analogous result holds if A is strictly column diagonally dominant (defined analogously).

As we will see in some numerical examples, the convergence of the Jacobi method is usually rather slow. A (usually) faster method is discussed next.

The Gauss-Seidel method

To see how the Jacobi method can be improved we consider an example.

Example For the system

$$\begin{aligned} 2x_1 + x_2 &= 6 \\ x_1 + 2x_2 &= 6 \end{aligned}$$

or

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

the Jacobi method looks like

$$\begin{aligned} x_1^{(k)} &= \left(6 - x_2^{(k-1)}\right)/2 \\ x_2^{(k)} &= \left(6 - x_1^{(k-1)}\right)/2. \end{aligned}$$

The Gauss-Seidel method

$$\begin{aligned} x_1^{(k)} &= \left(6 - x_2^{(k-1)}\right)/2 \\ x_2^{(k)} &= \left(6 - x_1^{(k-1)}\right)/2. \end{aligned}$$

In order to obtain an improvement we notice that the value of $x_1^{(k-1)}$ used in the second equation is actually outdated since we already computed a newer version, $x_1^{(k)}$, in the first equation. Therefore, we might consider

$$\begin{aligned} x_1^{(k)} &= \left(6 - x_2^{(k-1)}\right)/2 \\ x_2^{(k)} &= \left(6 - x_1^{(k)}\right)/2 \end{aligned}$$

instead. This is known as the *Gauss-Seidel method*. The general algorithm is of the form

The Gauss-Seidel method

Algorithm (Gauss-Seidel method)

```
Let $\boldsymbol{x}^{(0)}$ be an arbitrary initial guess
for $k = 1, 2, \ldots$
    for $i = 1 : m$
        $x_i^{(k)} = \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{m} a_{ij} x_j^{(k-1)} \right) / a_{ii}$
    end
end
```

The Gauss-Seidel method

Example For one step of the finite difference solution of the Poisson problem we get

```
for $i = 1 : n - 1$
    for $j = 1 : n - 1$
        $u_{i,j}^{(k)} = \left( u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k-1)} + u_{i,j+1}^{(k-1)} + \frac{f_{i,j}}{n^2} \right) / 4$
    end
end
```

Note that the unknowns are now u_{ij} instead of x_i. This algorithm can be implemented in one line of Matlab

The Gauss-Seidel method

Remark Note that the implementation of the Gauss-Seidel algorithm for this example depends on the ordering of the grid points. We used the natural (or typewriter) ordering, i.e., we scan the grid points row by row from left to right. Sometimes a red-black (or chessboard) ordering is used. This is especially useful if the Gauss-Seidel method is to be parallelized.

In order to understand the matrix formulation of the Gauss-Seidel method in the spirit of (2) we again assume that $A = L + D + U$.

Now the splitting matrices are chosen as

$$\begin{aligned} M &= D + L \\ N &= -U, \end{aligned}$$

and (2) becomes

$$\begin{aligned} & M\boldsymbol{x}^{(k)} = N\boldsymbol{x}^{(k-1)} + \boldsymbol{b} \\ \iff \quad & (D+L)\boldsymbol{x}^{(k)} = \boldsymbol{b} - U\boldsymbol{x}^{(k-1)} \end{aligned}$$

The Gauss-Seidel method

$$\begin{aligned} & M\boldsymbol{x}^{(k)} = N\boldsymbol{x}^{(k-1)} + \boldsymbol{b} \\ \iff \quad & (D+L)\boldsymbol{x}^{(k)} = \boldsymbol{b} - U\boldsymbol{x}^{(k-1)} \end{aligned}$$

or

$$\boldsymbol{x}^{(k)} = (D+L)^{-1}\left(\boldsymbol{b} - U\boldsymbol{x}^{(k-1)}\right).$$

Note that this is also equivalent to

$$D\boldsymbol{x}^{(k)} = \boldsymbol{b} - L\boldsymbol{x}^{(k)} - U\boldsymbol{x}^{(k-1)}$$

or

$$\boldsymbol{x}^{(k)} = D^{-1}\left(\boldsymbol{b} - L\boldsymbol{x}^{(k)} - U\boldsymbol{x}^{(k-1)}\right).$$

This latter formula corresponds nicely to the algorithm above.

The convergence criteria for Gauss-Seidel iteration are a little more general than those for the Jacobi method. One can show

The Gauss-Seidel method

Theorem *The Gauss-Seidel method converges for any initial guess $\boldsymbol{x}^{(0)}$ if*

1. *A is strictly diagonally dominant, or*
2. *A is symmetric positive definite.*

Remark For a generic problem the Gauss-Seidel method converges faster than the Jacobi method. However, this does not mean that sometimes the Jacobi method may not be faster.

Remark A careful reader may notice that the sufficient conditions given in the convergence theorems for the Jacobi and Gauss-Seidel methods do not cover the matrix for our finite-difference Poisson problem. However, there are variations of the theorems that do cover this important example.

Successive Over-Relaxation (SOR)

One can accelerate the convergence of the Gauss-Seidel method by using a weighted average of the new Gauss-Seidel value with the one obtained during the previous iteration:

$$\boldsymbol{x}^{(k)} = (1-\omega)\boldsymbol{x}^{(k-1)} + \omega\boldsymbol{x}^{(k)}_{\mathrm{GS}}.$$

Here ω is the so-called *relaxation parameter*. If $\omega = 1$ we simply have the Gauss-Seidel method. For $\omega > 1$ one speaks of *over-relaxation*, and for $\omega < 1$ of *under-relaxation*.

The resulting algorithm known as *successive over-relaxation* (SOR) and is (obviously) a variation of the Gauss-Seidel algorithm.

Successive Over-Relaxation (SOR)

Algorithm (SOR)

```
Let $\boldsymbol{x}^{(0)}$ be an arbitrary initial guess
for $k = 1, 2, \ldots$
    for $i = 1 : m$
        $x_i^{(k)} = (1-\omega)x_i^{(k-1)} + \omega \underbrace{\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{m} a_{ij}x_j^{(k-1)}\right)/a_{ii}}_{=x_{i,\mathrm{GS}}^{(k)}}$
    end
end
```

Successive Over-Relaxation (SOR)

Example For one step of the SOR algorithm applied to the finite difference solution of the Poisson problem we get

```
for $i = 1 : n-1$
    for $j = 1 : n-1$
        $u_{i,j}^{(k)} = (1-\omega)u_{i,j}^{(k-1)} + \omega \left( u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k-1)} + u_{i,j+1}^{(k-1)} + \frac{f_{i,j}}{n^2} \right) /4$
    end
end
```

Remark Just as for the Gauss-Seidel algorithm, the implementation of the SOR method depends on the ordering of the grid points.

Successive Over-Relaxation (SOR)

To describe the SOR method in terms of splitting matrices we again assume that $A = L + D + U$, and take

$$\begin{aligned} M &= \frac{1}{\omega} D + L \\ N &= \left(\frac{1}{\omega} - 1\right) D - U. \end{aligned}$$

With these choices (2) becomes

$$\begin{aligned} & M\boldsymbol{x}^{(k)} = N\boldsymbol{x}^{(k-1)} + \boldsymbol{b} \\ \Longleftrightarrow \quad & \left(\frac{1}{\omega} D + L\right) \boldsymbol{x}^{(k)} = \left[\left(\frac{1}{\omega} - 1\right) D - U\right] \boldsymbol{x}^{(k-1)} + \boldsymbol{b}. \end{aligned}$$

Successive Over-Relaxation (SOR)

The rearrangements that show that this formulation is indeed equivalent to the formula used in the algorithm above are:

$$
\begin{aligned}
& \left(\frac{1}{\omega}D + L\right)\boldsymbol{x}^{(k)} = \left[\left(\frac{1}{\omega} - 1\right)D - U\right]\boldsymbol{x}^{(k-1)} + \boldsymbol{b} \\
\iff \quad & \frac{1}{\omega}D\boldsymbol{x}^{(k)} = \left[\left(\frac{1}{\omega} - 1\right)D - U\right]\boldsymbol{x}^{(k-1)} + \boldsymbol{b} - L\boldsymbol{x}^{(k)} \\
\iff \quad & \boldsymbol{x}^{(k)} = \omega D^{-1}\left[\left(\frac{1}{\omega} - 1\right)D - U\right]\boldsymbol{x}^{(k-1)} + \omega D^{-1}\boldsymbol{b} - \omega D^{-1}L\boldsymbol{x}^{(k)} \\
\iff \quad & \boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} - \omega\boldsymbol{x}^{(k-1)} - \omega D^{-1}U\boldsymbol{x}^{(k-1)} + \omega D^{-1}\boldsymbol{b} - \omega D^{-1}L\boldsymbol{x}^{(k)} \\
\iff \quad & \boldsymbol{x}^{(k)} = (1-\omega)\,\boldsymbol{x}^{(k-1)} - \omega\left[D^{-1}\left(\boldsymbol{b} - L\boldsymbol{x}^{(k)} - U\boldsymbol{x}^{(k-1)}\right)\right].
\end{aligned}
$$

Note that the expression inside the square brackets on the last line is just what we had for the Gauss-Seidel method earlier.

Successive Over-Relaxation (SOR)

One can prove a general convergence theorem that is similar to those for the Jacobi and Gauss-Seidel methods:

Theorem *If A is symmetric positive definite then the SOR method with $0 < \omega < 2$ converges for any starting value $\boldsymbol{x}^{(0)}$.*

Remark Note that this theorem says nothing about the speed of convergence. In fact, finding a good value for the relaxation parameter ω is quite difficult. The value of ω that yields the fastest convergence of the SOR method is known only in very special cases. For example, if A is tridiagonal then

$$\omega_{\text{opt}} = \frac{2}{1 + \sqrt{1 - \rho(G)}},$$

where $G = M^{-1}N = (D + L)^{-1}(-U)$ is the iteration matrix for the Gauss-Seidel method.