6. Iterative Methods for Linear Systems

*The stepwise approach to the solution*...
6.1. Large Sparse Systems of Linear Equations I – Relaxation Methods

Introduction

- Systems of linear equations, which have to be solved numerically, often stem from the discretization of ordinary (for boundary value problems see chapter 8) or partial differential equations. The **direct** solving methods studied in chapter 5 usually are not applicable:
  - First, $n$ most times is so big (usually $n$ directly correlates with the number of grid points, which leads to very big $n$, especially for unsteady partial differential equations (three spatial and one time variable)), such that a computational cost of the complexity $O(n^3)$ is not acceptable.
  - Second, such matrices are usually **sparsely populated** (i.e. $O(1)$ or $O(\log n)$ non-zero values per row, e.g.) and have a certain **structure** (tridiagonal, band structure, block structure etc.), which, of course, has a positive effect on memory and computing time; methods of elimination often destroy this structure and, therefore, undo those structural advantages.
Example for a sparse matrix that is filled during elimination (’*’
denotes a non-zero matrix entry):

\[
\begin{pmatrix}
  * & * & \cdots & * & * \\
  * & * & 0 & \cdots & 0 \\
  \vdots & 0 \ddots & \ddots & \vdots \\
  \vdots & \vdots \ddots & \ddots & 0 \\
  * & 0 & \cdots & 0 & * \\
\end{pmatrix}
\]

After elimination:

\[
\begin{pmatrix}
  * & * & \cdots & \cdots & * \\
  * & * & \cdots & \cdots & * \\
  \vdots & \vdots & \ddots & \ddots & \vdots \\
  \vdots & \vdots & \ddots & \ddots & \vdots \\
  * & * & \cdots & \cdots & * \\
\end{pmatrix}
\]
For large and sparse matrices or systems of linear equations, therefore, we prefer **iterative** methods:

- Those start with an *initial approximation* $x^{(0)}$ and generate a sequence of approximate values $x^{(i)}$, $i = 1, 2, \ldots$, which, in case of convergence, converges to the exact solution $x$.

- One iteration step typically costs $O(n)$ arithmetic operations in case of a sparse matrix (less is hardly possible if you want to update every vector component of $x^{(i)}$ in every step). Therefore, the construction of iterative algorithms depends on how many iteration steps are required to obtain a certain accuracy.
Overview of Relaxation Methods

- The probably oldest iterative methods for solving systems of linear equations $Ax = b$ with $A \in \mathbb{R}^{n,n}$ and $x, b \in \mathbb{R}^n$ are the so-called relaxation or smoothing methods:
  - the Richardson method,
  - the Jacobi method,
  - the Gauss-Seidel method as well as
  - SOR, the successive over-relaxation.

- Basic principles:
  - For all methods mentioned, the starting point is the residual $r^{(i)}$ already introduced,
    \[
    r^{(i)} := b - Ax^{(i)} = Ax - Ax^{(i)} = A(x - x^{(i)}) = -Ae^{(i)},
    \]
    where $x^{(i)}$ is the current approximation for the exact solution $x$ after $i$ iteration steps and $e^{(i)}$ denotes the respective error.
  - As $e^{(i)}$ is not available (the error can not be determined if the exact solution $x$ is unknown), due to the relation above, it turns out to be reasonable to take the vector $r^{(i)}$ as direction in which we want to look for an update of $x^{(i)}$. 
The Richardson method directly takes the residual as adjustment for $x^{(i)}$. The Jacobi and the Gauss-Seidel method try harder. Their idea to adjust the $k$-th component of $x^{(i)}$ is to eliminate $r_k^{(i)}$. The SOR method and its counterpart, the **damped** relaxation, additionally take into account that such an update often either overshoots the mark or is not sufficient.
Important Methods of Relaxation

- **Richardson iteration:**
  
  \[
  \text{for } i = 0, 1, \ldots \\
  \text{for } k = 1, \ldots, n: \quad x_k^{(i+1)} := x_k^{(i)} + r_k^{(i)}
  \]

  Here, the residual \( r^{(i)} \) is simply used componentwise as update to adjust the active approximation \( x^{(i)} \).

- **Jacobi iteration:**
  
  \[
  \text{for } i = 0, 1, \ldots \\
  \text{for } k = 1, \ldots, n: \quad y_k := \frac{1}{a_{kk}} \cdot r_k^{(i)} \\
  \text{for } k = 1, \ldots, n: \quad x_k^{(i+1)} := x_k^{(i)} + y_k
  \]

  – In every substep \( k \) of a step \( i \), an update \( y_k \) is computed and stored.
  – Applied immediately, this would lead to the (momentary) disappearance of the \( k \)-th component of the residual \( r^{(i)} \) (easy to verify by inserting).
  – With this current approximation for \( x \), equation \( k \) would therefore be solved exactly – an improvement that would be lost, of course, in the following substep for the equation \( k + 1 \).
  – However, these updates of a component are not carried out immediately, but only at the end of an iteration step (second \( k \)-loop).
Important Methods of Relaxation (2)

- **Gauss-Seidel iteration:**
  
  \[
  \begin{align*}
  r_k^{(i)} &:= b_k - \sum_{j=1}^{k-1} a_{kj} x_j^{(i+1)} - \sum_{j=k}^{n} a_{kj} x_j^{(i)} \\
  y_k &:= \frac{1}{a_{kk}} \cdot r_k^{(i)} , \quad x_k^{(i+1)} := x_k^{(i)} + y_k
  \end{align*}
  \]

  - In principle, the update calculated here is at most the same as in the Jacobi method. However, the update is not performed at the end of the iteration step, but always immediately.
  
  - Therefore, the new modified values for the components 1 to \(k-1\) are already available for the update of component \(k\).

- Sometimes, a **damping** (multiplying the update with a factor \(0 < \alpha < 1\)) or an **over-relaxation** (factor \(1 < \alpha < 2\)) leads to better convergence behavior for each of the three methods outlined:

  \[
  x_k^{(i+1)} := x_k^{(i)} + \alpha y_k
  \]

  - In the case of Gauss-Seidel method, the version with \(\alpha > 1\) is mainly used. It is denoted as **SOR method** (**successive over-relaxation**).
  
  - In the case of Jacobi method, in contrary, damping mostly is in use.
Gauss-Seidel – Schematic Principle of one Iteration

Step 1:
\[
\begin{pmatrix}
 a_{11} & a_{12} & \cdots & a_{1n} \\
 a_{21} & a_{22} & \cdots & a_{2n} \\
 \vdots & \vdots & \ddots & \vdots \\
 a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 \vdots \\
 x_n
\end{pmatrix}
= 
\begin{pmatrix}
 b_1 \\
 b_2 \\
 \vdots \\
 b_n
\end{pmatrix}
\]
← solve for \( x_1 \)

Step 2:
\[
\begin{pmatrix}
 a_{11} & a_{12} & \cdots & a_{1n} \\
 a_{21} & a_{22} & \cdots & a_{2n} \\
 \vdots & \vdots & \ddots & \vdots \\
 a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 \vdots \\
 x_n
\end{pmatrix}
= 
\begin{pmatrix}
 b_1 \\
 b_2 \\
 \vdots \\
 b_n
\end{pmatrix}
\]
← solve for \( x_2 \)

\vdots

Step \( n \):
\[
\begin{pmatrix}
 a_{11} & a_{12} & \cdots & a_{1n} \\
 a_{21} & a_{22} & \cdots & a_{2n} \\
 \vdots & \vdots & \ddots & \vdots \\
 a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 \vdots \\
 x_n
\end{pmatrix}
= 
\begin{pmatrix}
 b_1 \\
 b_2 \\
 \vdots \\
 b_n
\end{pmatrix}
\]
← solve for \( x_n \)
Jacobi – Schematic Principle of one Iteration

Step 1:
\[
\begin{pmatrix}
 a_{11} & a_{12} & \cdots & a_{1n} \\
 a_{21} & a_{22} & \cdots & a_{2n} \\
 \vdots & \vdots & \ddots & \vdots \\
 a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 \vdots \\
 x_n
\end{pmatrix}
= 
\begin{pmatrix}
 b_1 \\
 b_2 \\
 \vdots \\
 b_n
\end{pmatrix}
\leftarrow \text{solve for } x_1 \Rightarrow y_1
\]

Step 2:
\[
\begin{pmatrix}
 a_{11} & a_{12} & \cdots & a_{1n} \\
 a_{21} & a_{22} & \cdots & a_{2n} \\
 \vdots & \vdots & \ddots & \vdots \\
 a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 \vdots \\
 x_n
\end{pmatrix}
= 
\begin{pmatrix}
 b_1 \\
 b_2 \\
 \vdots \\
 b_n
\end{pmatrix}
\leftarrow \text{solve for } x_2 \Rightarrow y_2
\]

\vdots

Step \( n \):
\[
\begin{pmatrix}
 a_{11} & a_{12} & \cdots & a_{1n} \\
 a_{21} & a_{22} & \cdots & a_{2n} \\
 \vdots & \vdots & \ddots & \vdots \\
 a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 \vdots \\
 x_n
\end{pmatrix}
= 
\begin{pmatrix}
 b_1 \\
 b_2 \\
 \vdots \\
 b_n
\end{pmatrix}
\leftarrow \text{solve for } x_n \Rightarrow y_n
\]

After the iteration, copy values in the intermediate data structure \( \vec{y} \) to \( \vec{x} \).
Discussion: Additive Decomposition of the System Matrix

- In order to quickly analyze the convergence of the methods above, we need an algebraic formulation (instead of the algorithmic one).
- Every approach shown is based on the simple idea of writing the matrix $A$ as a sum
  \[ A = M + (A - M), \]
  where $Mx = b$ is quite simple to solve and the difference $A - M$ should not be too big with regard to some matrix norm.
- With the help of such an adequate $M$, we will be able to write the Richardson, Jacobi, Gauss-Seidel, and SOR methods as
  \[ Mx^{(i+1)} + (A - M)x^{(i)} = b \]
or, solved for $x^{(i+1)}$, as:
  \[ x^{(i+1)} := M^{-1}b - M^{-1}(A - M)x^{(i)} = M^{-1}b - (M^{-1}A - I)x^{(i)} = x^{(i)} + M^{-1}r^{(i)}. \]
Furthermore, we decompose $A$ additively in its diagonal part $D_A$, its strict lower triangular part $L_A$ as well as its strict upper triangular part $U_A$:

$$A = L_A + D_A + U_A.$$ 

With this, we can show the following relations:

- **Richardson:** $M := I$,
- **Jacobi:** $M := D_A$,
- **Gauss-Seidel:** $M := D_A + L_A$,
- **SOR:** $M := \frac{1}{\alpha} D_A + L_A$.

Considering the algorithmic formulation of the Richardson as well as that of the Jacobi method, the first two equations are obvious:

- At Richardson, the residual is directly used as update. Therefore, the identity $I$ results as the prefactor $M$.
- At Jacobi, the residual is divided by the diagonal element. Therefore, the inverse of the diagonal part $D_A$ results as the prefactor $M$. 

As the Gauss-Seidel iteration is a special case of the SOR method \((\alpha = 1)\), it's sufficient to prove the formula above for \(M\) in the general SOR case. From the algorithm, it follows immediately

\[
x_k^{(i+1)} := x_k^{(i)} + \alpha \left( b_k - \sum_{j=1}^{k-1} a_{kj} x_j^{(i+1)} - \sum_{j=k}^{n} a_{kj} x_j^{(i)} \right) / a_{kk}
\]

\[
\Leftrightarrow x^{(i+1)} := x^{(i)} + \alpha D_A^{-1} \left( b - L_A x^{(i+1)} - (D_A + U_A) x^{(i)} \right)
\]

\[
\Leftrightarrow \frac{1}{\alpha} D_A x^{(i+1)} = \frac{1}{\alpha} D_A x^{(i)} + b - L_A x^{(i+1)} - (D_A + U_A) x^{(i)}
\]

\[
\Leftrightarrow \left( \frac{1}{\alpha} D_A + L_A \right) x^{(i+1)} + \left( (1 - \frac{1}{\alpha}) D_A + U_A \right) x^{(i)} = b
\]

\[
\Leftrightarrow M x^{(i+1)} + (A - M) x^{(i)} = b,
\]

which proves the statement for the SOR method.
Discussion: Convergence

- As far as convergence is concerned, two immediate consequences follow from the approach

\[ Mx^{(i+1)} + (A - M)x^{(i)} = b \]  

- If the sequence \( (x^{(i)}) \) is convergent, then the exact solution \( x \) of our system \( Ax = b \) is the limit.

- For the analysis, assume that the iteration matrix \(-M^{-1}(A - M)\) (i.e. the matrix that is applied to \( e^{(i)} \) to get \( e^{(i+1)} \); see below) is symmetric. Then the spectral radius \( \rho \) (i.e. the eigenvalue with the biggest absolute value) is the decisive parameter for convergence:

\[ \left( \forall x^{(0)} \in \mathbb{R}^n : \lim_{i \to \infty} x^{(i)} = x = A^{-1}b \right) \iff \rho < 1. \]

To see that, subtract \( Mx + (A - M)x = b \) from the equation at the very top:

\[ Me^{(i+1)} + (A - M)e^{(i)} = 0 \iff e^{(i+1)} = -M^{-1}(A - M)e^{(i)}. \]
Thus, we get

\[
\lim_{i \to \infty} \| e^{(i)} \| \leq \lim_{i \to \infty} \| (-M^{-1}(A - M))^i \| \cdot \| e^{(0)} \| = \lim_{i \to \infty} \rho^i \cdot \| e^{(0)} \|
\]

for any norm \( \| \cdot \| \) with the spectral radius \( \rho \) of the iteration matrix \(-M^{-1}(A - M)\). If all absolute values of the eigenvalues are smaller than 1 and, therefore, \( \rho < 1 \) holds, all error components are reduced in the limit (not necessarily in every iteration step!). In case of \( \rho > 1 \), at least one error component will build up.
The goal of the construction of iteration matrix must of course be a spectral radius of the iterative matrix that is as small as possible (as close to zero as possible).

- There are a number of results for the convergence of the different methods of which we should mention a few important ones:
  - For the convergence of the SOR method, $0 < \alpha < 2$ is necessary.
  - If $A$ is positive definite, the SOR method (for $0 < \alpha < 2$) as well as the Gauss-Seidel iteration are convergent.
  - If $A$ and $2D_A - A$ are both positive definite, the Jacobi method converges.
  - If $A$ is strictly diagonally dominant (i.e. $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ for all $i$), then the Jacobi and the Gauss-Seidel method are convergent.
  - In certain cases, the ideal parameter $\alpha$ can be determined ($\rho$ minimal, such that the error reduction per iteration step is maximal).

- The Gauss-Seidel iteration is not generally better than the Jacobi method (as you might assume because of the update performed immediately). There are examples in which the former converges and the latter diverges or the other way round. However, in many cases, the Gauss-Seidel method does with only half of the iteration steps of the Jacobi method.
The Spectral Radius of Typical Iterative Matrices

- Obviously, $\rho$ is not only decisive for the question whether the iteration converges at all, but also for its quality, i.e. its speed of convergence: The smaller $\rho$ is, the faster all components of the error $e^{(i)}$ are reduced in every iteration step.

- In practice, the results about convergence above unfortunately have rather theoretical value because $\rho$ is often so close to 1 that – in spite of convergence – the number of iteration steps necessary to reach a sufficient accuracy is way too big.

- An important sample scenario is the discretization of partial differential equations:
  
  - It is characteristic that $\rho$ depends on the dimension $n$ of the problem and, therefore, on the resolution $h$ of the underlying grid, i.e. for example

    \[ \rho = O(1 - h^2_l) = O \left(1 - \frac{1}{4^l}\right) \]

    with the mesh width $h_l = 2^{-l}$.

    - This is a huge disadvantage: The finer and, therefore, the more precise our grid is, the more miserable the behavior of convergence of our iterative methods gets. Therefore, better iterative solvers are a must (e.g. multigrid methods, cf. next section)!
Discussion: Other Aspects

- For Richardson and Jacobi iterations, the order of unknowns in $\vec{x}$ does not matter (all new values in $\vec{x}^{i+1}$ only depend on old values from $\vec{x}^i$).
  - This requires the storage of intermediate values (in $\vec{r}$ or $\vec{y}$), but
  - allows for the parallel execution of all updates in $\vec{x}$.

- For Gauss-Seidel and SOR, the convergence properties depend on the order of unknowns in $\vec{x}$.
  - Thus, no intermediate storage is required, but
  - both iterations have to be executed sequentially (following the order of unknowns in $\vec{x}$).

- In the second semester (Scientific Computing II), parallel versions of Gauss-Seidel are discussed for a special types of linear systems resulting from the discretization of partial differential equations.