

• Numerical Example

Let’s start with a numerical example which we’ll use to illustrate some of the ideas.

The example occurs in the solution of PDEs which we have not discussed. However, it can be motivated physically. Suppose you have a rectangular metal plate where the temperature along the boundary depends on the location on the boundary, but is constant in time. You want to approximate the temperature distribution in the interior of the plate. To that end you define a rectangular grid and require that the temperature approximation at each interior grid point equal the mean of the temperatures at its four neighbors.

Of course this situation could be generalized. You might have a three-dimensional object like the block of an internal combustion engine, and the temperature on the boundary might depend on time as well as location.

Returning to a two-dimensional rectangle, let’s suppose our grid consists of points \((x_i, y_j)\) where

\[ x_i = y_i = ih, \quad i = 0 \ldots N \quad \text{and} \quad h = \frac{1}{N} \]

Let \(U_{ij}\) denote the approximate temperature at \((x_i, y_j)\). Then we get the \((N - 1)^2\) equations

\[ 4U_{ij} - U_{i+1,j} - U_{i-1,j} - U_{i,j+1} - U_{i,j-1} = 0, \quad i, j = 1, \ldots, N-1 \]
in as many unknowns.

- For example, in the case $N = 5$ we get the (extremely
small) grid shown in Figure 1 and the matrix

\[
\begin{pmatrix}
\end{pmatrix}
\]

(2)

where

\[ C = -4. \]

- We want to think of the grid, and the matrix, as being much larger. For example, if

\[ N = 101, \]

a realistic choice, then the matrix, and the linear system, would be \(10,000 \times 10,000\). This would not be particularly large by today’s standards.

- Think about the structure of this matrix!
- symmetric
- eigenvalues (-8, 0)
- negative definite $Ax = b$ neg. def
- $A^2 = -b$ pos. def
- sparse mostly zero
- don't need to store $A$
Iterative Solution of Large Linear Systems

• Next let’s generalize the context somewhat and think about the solution of general sparse systems.

• A matrix is sparse if so many of its entries are zero that it pays to utilize that fact.

• Consider

\[ Ax = b \]  

(3)

where \( A \) is a \textbf{sparse} \( n \times n \) matrix.

• The basic idea of designing iterative methods for the solution of (3) is to solve the \( i \)-th equation

\[ \sum_{j=1}^{n} a_{ij} x_j = b_i \]

(4)

for the \( i \)-th unknown, and then turning the resulting equation into an iterative method:

• Solving (4) for \( x_i \) gives

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

• Suppose now that we are given some approximation \( x^{[k]} \) of the solution of (3). In the \textbf{Jacobi} or \textbf{Gauss-Jacobi} Method we define a new approximation \( x^{[k+1]} \) by

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

(5)

where \( x^{[0]} \) is some given initial vector and \( k = 0, 1, 2, \ldots \).
• The choice of the starting point $x^{[0]}$ depends of course on the context.

• It should be clear how the iteration (5) takes into account the sparsity of the matrix $A$. In the summations on the right, we would skip all terms for which $a_{ij} = 0$.

• It seems natural to use new approximations as soon as they become available.

• This gives rise to the **Gauss-Seidel** Method.

• In terms of the general system (3) we obtain

$$
x_i^{[k+1]} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k+1]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right) \tag{6}
$$

• It would seem that the Gauss-Seidel Method is clearly superior over the Jacobi Method.

### Successive Overrelaxation

• The **Successive Overrelaxation Method** (or SOR for short) defines the new approximation as a linear combination of the old approximation and the Gauss-Seidel Approximation. For the general system $Ax = b$ we define

$$
x_i^{[k+1]} = (1-\omega)x_i^{[k]} + \frac{\omega}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k+1]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right) \tag{7}
$$

• The constant $\omega$ is called the **relaxation parameter**. It is usually in the interval $(1, 2)$. If it is in the interval $(0, 1)$ the process is called **underrelaxation**.
Matrix Formulation

• We now ask the following questions:
  − Do these methods converge?
  − If so, how fast?
  − What are good values, or the best value, of $\omega$.

• To answer these question we start by expression the methods in terms of matrices.

• Return to the general system (3) and write

$$A = L + D + U$$

where $D$ is diagonal and $L$ and $U$ are strictly lower and upper triangular, respectively.

• For example,

$$\begin{bmatrix}
-4 & 1 & 1 \\
1 & -4 & 1 \\
1 & 1 & -4
\end{bmatrix} = \begin{bmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
1 & 1 & 0
\end{bmatrix} + \begin{bmatrix}
-4 & 0 & 0 \\
0 & -4 & 0 \\
0 & 0 & -4
\end{bmatrix} + \begin{bmatrix}
0 & 1 & 1 \\
0 & 0 & 1 \\
0 & 0 & 0
\end{bmatrix}$$

Jacobi

• In terms of our matrix partition the Jacobi Iteration (5)

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

(8)

turns into

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

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

$$= T_Jx^{[k]} + c_J$$
where

\[ T_j = -D^{-1}(L + U) \quad \text{and} \quad c_j = D^{-1}b. \]

- Similarly, the Gauss-Seidel Iteration (6)

\[
\begin{align*}
  x_i^{[k+1]} &= \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k+1]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right) \\
  &= T_{GS} x_i^{[k]} + c_{GS}
\end{align*}
\]

becomes

\[ x^{[k+1]} = D^{-1}(b - Lx^{[k+1]} - Ux^{[k]}) \]

which can be rewritten as

\[ x^{[k+1]} = T_{GS}x^{[k]} + c_{GS} \]

where

\[ T_{GS} = -(D + L)^{-1}U \quad \text{and} \quad c_{GS} = (D + L)^{-1}b. \]

- Similarly, the SOR iteration (7)

\[
\begin{align*}
  x_i^{[k+1]} &= (1-\omega)x_i^{[k]} + \frac{\omega}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k+1]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right) \\
  &= T_\omega x_i^{[k]} + c_\omega
\end{align*}
\]

(9)

can be rewritten as

\[ x^{[k+1]} = (1 - \omega)x^{[k]} + \omega D^{-1} \left( b - Lx^{[k+1]} - Ux^{[k]} \right) \]

which can be rewritten as

\[ x^{[k+1]} = T_\omega x^{[k]} + c_\omega \]

where

\[ T_\omega = (D + \omega L)^{-1} [(1 - \omega)D - \omega U] \]
and
\[ c_\omega = \omega (D + \omega L)^{-1}b. \]

- Norms can be used to define and analyze the convergence of a sequence of vectors. We say that the sequence 
\[ x^{(0)}, x^{(1)}, x^{(2)}, x^{(3)}, \ldots \]
converges to \( x \) if
\[ \lim_{k \to \infty} \| x^{(k)} - x \| = 0. \]

- Because of the equivalence of norms the convergence of the sequence is independent of the choice of the norm used for the analysis.

- Recall the fixed point iteration
\[ Ax = b \iff x = Tx + c \quad (10) \]
leading to
\[ x^{[0]} \text{ given,} \quad x^{[k+1]} = Tx^{[k]} + c, \quad k = 1, \ldots \quad (11) \]

- Defining the error
\[ e^{[k]} = x - x^{[k]} \]
and subtracting (11) from (10) we get
\[ e^{[k+1]} = Te^{[k]} \]
and, for any induced matrix norm
\[ \| e^{[k+1]} \| \leq \| T \| \| e^{[k]} \| \]

- Thus the iteration will converge if we can find an induced matrix norm such the
\[ \| T \| < 1. \]

- Moreover, since the spectral radius can be approximated arbitrarily closely by an induced matrix norm we would like
\[ \rho(T) < 1 \]
and, for the SOR method we would like \( \omega \) to minimize the spectral radius of \( T(\omega) \).
• For the special case of our three iterative methods applied to the two dimensional Poisson equation the spectral radii can actually computed. This leads to some beautiful mathematics, which, however, is beyond our scope. For a full discussion refer to the above mentioned book by Varga.

• We will, however, look at some of the results. Table 1 compares the performance of our three iteration methods on the two dimensional Poisson equation on an \( n \times n \) grid. The individual columns contain the following data:

- \( n \) : the number of interior grid points in one dimension. The total number of equations and unknowns is thus \( n^2 \). The step size is

\[
h = \frac{\pi}{n}
\]

and the domain of the PDE is \([0, \pi] \times [0, \pi]\). (The purpose of considering this square, rather than the unit square, is to avoid the repeated appearance of \( \pi \) in the formulas below.)

- \( \omega_b \) : The best choice of \( \omega \) for the SOR method, given by

\[
\omega_b = \frac{2}{1 + \sin h}.
\]

- \( \rho_J \) : the spectral radius of \( T_J \),

\[
\rho_J = \cos h.
\]

- \( n_J \) : The number of steps required by the Jacobi Method to reduce the error by a factor 10.

- \( \rho_{GS} \) : the spectral radius of \( T_{GS} \),

\[
\rho_{GS} = \cos^2 h = \rho_J^2.
\]

- \( n_{GS} \) : The number of steps required by the Gauss-Seidel Method to reduce the error by a factor 10.
\[ \rho_{SOR} \] The spectral radius of \( T_\omega \) for \( \omega = \omega_b \):

\[ \rho_\omega = \frac{\cos^2 h}{(1 + \sin h)^2}. \]

\[ n_{sor} \] The number of steps required by the SOR method, with the best choice of \( \omega \) to reduce the error by a factor 10.

- Note the following points:
  - The Gauss-Seidel Method does perform better than the Jacobi Method. In fact, it takes exactly one step for every two steps of the Jacobi Method.
  - As the discretization gets finer the best choice of \( \omega \) approaches 2.
  - The spectral radii of the Gauss-Seidel Method and the best SOR method are asymptotically equal (and approach 1 as \( h \) approaches zero), yet the reduction in the number of sweeps by the SOR method is substantially smaller than the number of steps required by the Gauss-Seidel method, and the advantage of the SOR method increases as \( h \) decreases.
It is also instructive to compare the SOR method with Gaussian Elimination. The columns of Table 2 contain the following data:

- **n** Same as in Table 1, the number of equations equals \( n^2 \).
- **SOR** The number of divisions (there are no multiplications) required by the SOR method to reduce the error by a factor 10.
- **Gauss** The number of multiplications and divisions to use Gaussian Elimination to solve a general \( n^2 \times n^2 \) linear system.
- **ratio** The quotient

\[
\frac{\text{Gauss}}{\text{SOR}}
\]

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \omega_b )</th>
<th>( \rho_J )</th>
<th>( n_J )</th>
<th>( \rho_{GS} )</th>
<th>( n_{GS} )</th>
<th>( \rho_{SOR} )</th>
<th>( n_{SOR} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1.61379</td>
<td>.97094</td>
<td>78</td>
<td>.94273</td>
<td>39</td>
<td>.61379</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>1.65575</td>
<td>.97815</td>
<td>104</td>
<td>.95677</td>
<td>52</td>
<td>.65575</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>1.68955</td>
<td>.98297</td>
<td>134</td>
<td>.96624</td>
<td>67</td>
<td>.68955</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>1.72945</td>
<td>.98769</td>
<td>185</td>
<td>.97553</td>
<td>92</td>
<td>.72945</td>
<td>7</td>
</tr>
<tr>
<td>22</td>
<td>1.76031</td>
<td>.99069</td>
<td>246</td>
<td>.98146</td>
<td>123</td>
<td>.76031</td>
<td>8</td>
</tr>
<tr>
<td>26</td>
<td>1.79197</td>
<td>.99324</td>
<td>339</td>
<td>.98652</td>
<td>169</td>
<td>.79197</td>
<td>9</td>
</tr>
<tr>
<td>31</td>
<td>1.82147</td>
<td>.99518</td>
<td>477</td>
<td>.99039</td>
<td>238</td>
<td>.82147</td>
<td>11</td>
</tr>
<tr>
<td>37</td>
<td>1.84744</td>
<td>.99658</td>
<td>673</td>
<td>.99318</td>
<td>336</td>
<td>.84744</td>
<td>13</td>
</tr>
<tr>
<td>44</td>
<td>1.86958</td>
<td>.99756</td>
<td>944</td>
<td>.99513</td>
<td>472</td>
<td>.86958</td>
<td>16</td>
</tr>
<tr>
<td>52</td>
<td>1.88815</td>
<td>.99824</td>
<td>1309</td>
<td>.99649</td>
<td>654</td>
<td>.88815</td>
<td>19</td>
</tr>
<tr>
<td>62</td>
<td>1.90504</td>
<td>.99876</td>
<td>1851</td>
<td>.99752</td>
<td>925</td>
<td>.90504</td>
<td>23</td>
</tr>
<tr>
<td>74</td>
<td>1.91961</td>
<td>.99912</td>
<td>2623</td>
<td>.99825</td>
<td>1311</td>
<td>.91961</td>
<td>27</td>
</tr>
<tr>
<td>88</td>
<td>1.93182</td>
<td>.99938</td>
<td>3695</td>
<td>.99875</td>
<td>1847</td>
<td>.93182</td>
<td>32</td>
</tr>
<tr>
<td>105</td>
<td>1.94244</td>
<td>.99956</td>
<td>5241</td>
<td>.99912</td>
<td>2620</td>
<td>.94244</td>
<td>38</td>
</tr>
<tr>
<td>126</td>
<td>1.95173</td>
<td>.99969</td>
<td>7525</td>
<td>.99939</td>
<td>3762</td>
<td>.95173</td>
<td>46</td>
</tr>
<tr>
<td>151</td>
<td>1.95950</td>
<td>.99979</td>
<td>10779</td>
<td>.99957</td>
<td>5389</td>
<td>.95950</td>
<td>55</td>
</tr>
<tr>
<td>181</td>
<td>1.96606</td>
<td>.99985</td>
<td>15454</td>
<td>.99970</td>
<td>7727</td>
<td>.96606</td>
<td>66</td>
</tr>
<tr>
<td>217</td>
<td>1.97159</td>
<td>.99990</td>
<td>22173</td>
<td>.99979</td>
<td>11086</td>
<td>.97159</td>
<td>79</td>
</tr>
</tbody>
</table>

**Table 1:** comparison of J, GS, SOR.
### Table 2: comparison of SOR and Gauss E.

- The comparison is a bit unfair to Gaussian Elimination since it does not take into account sparsity. However, any elimination process that requires an effort that grows cubically with the number of equations would give similar results. The ratios would be as in the table, except they would be reduced by a constant factor. Similarly, considering reductions of the error by a factor other than 10 would alter the ratios only by a constant factor.

- It is clear that for large systems the SOR method outperforms Gaussian Elimination.

- Our final comment concerns the Jacobi Method. It seems intuitively clear, and is true in our particular example, that the Gauss-Seidel Method outperforms the Jacobi Method. However, the situation changes when you consider the ability to compute in parallel. As an extreme case, con-

```
<table>
<thead>
<tr>
<th>n</th>
<th>SOR</th>
<th>Gauss</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>5.76E+02</td>
<td>9.95E+05</td>
<td>1.73E+03</td>
</tr>
<tr>
<td>14</td>
<td>9.80E+02</td>
<td>2.51E+06</td>
<td>2.56E+03</td>
</tr>
<tr>
<td>16</td>
<td>1.54E+03</td>
<td>5.59E+06</td>
<td>3.64E+03</td>
</tr>
<tr>
<td>19</td>
<td>2.53E+03</td>
<td>1.57E+07</td>
<td>6.21E+03</td>
</tr>
<tr>
<td>22</td>
<td>3.87E+03</td>
<td>3.78E+07</td>
<td>9.76E+03</td>
</tr>
<tr>
<td>26</td>
<td>6.08E+03</td>
<td>1.03E+08</td>
<td>1.69E+04</td>
</tr>
<tr>
<td>31</td>
<td>1.06E+04</td>
<td>2.96E+08</td>
<td>2.80E+04</td>
</tr>
<tr>
<td>37</td>
<td>1.78E+04</td>
<td>8.55E+08</td>
<td>4.81E+04</td>
</tr>
<tr>
<td>44</td>
<td>3.10E+04</td>
<td>2.42E+09</td>
<td>7.81E+04</td>
</tr>
<tr>
<td>52</td>
<td>5.14E+04</td>
<td>6.59E+09</td>
<td>1.28E+05</td>
</tr>
<tr>
<td>62</td>
<td>8.84E+04</td>
<td>1.89E+10</td>
<td>2.14E+05</td>
</tr>
<tr>
<td>74</td>
<td>1.48E+05</td>
<td>5.47E+10</td>
<td>3.70E+05</td>
</tr>
<tr>
<td>88</td>
<td>2.48E+05</td>
<td>1.55E+11</td>
<td>6.25E+05</td>
</tr>
<tr>
<td>105</td>
<td>4.19E+05</td>
<td>4.47E+11</td>
<td>1.07E+06</td>
</tr>
<tr>
<td>126</td>
<td>7.30E+05</td>
<td>1.33E+12</td>
<td>1.83E+06</td>
</tr>
<tr>
<td>151</td>
<td>1.25E+06</td>
<td>3.95E+12</td>
<td>3.15E+06</td>
</tr>
<tr>
<td>181</td>
<td>2.16E+06</td>
<td>1.17E+13</td>
<td>5.42E+06</td>
</tr>
<tr>
<td>217</td>
<td>3.72E+06</td>
<td>3.48E+13</td>
<td>9.36E+06</td>
</tr>
</tbody>
</table>
```
Consider a grid where each node is a (small) CPU. The Jacobi method can be implemented by having each grid point continually query its four neighbors about their temperature and update its own temperature accordingly. This would provide an effective method even if the CPUs are not synchronized!