1 GSW… Iterative Techniques for y = Ax

I'm going to cheat here. There are a lot of iterative techniques that can be used to solve the general case of a set of simultaneous equations (written in the matrix form as **y** = **Ax**), but this chapter isn't going to talk about all of the ones in this book. This chapter only talks about the Jacobi and Gauss-Seidel methods, and variations on them. For the steepest descent method and the related method of conjugate gradients, see the next chapter.

1.1 The Jacobi Iterative Method

The Jacobi method is an iterative technique for solving the square series of simultaneous equations **y** = **Ax**. The idea is to take the matrix **A**, and express it as the sum of a matrix of the diagonal elements **D**, and a matrix of all the other elements, **E**. For example:

$$\mathbf{A} = \begin{bmatrix} -2 & 1 & -1 \\ -1 & 2 & 1 \\ 2 & -1 & 4 \end{bmatrix} = \mathbf{D} + \mathbf{E} = \begin{bmatrix} -2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 2 & -1 & 0 \end{bmatrix} \tag{0.1}$$

It's very easy to invert a matrix that consists only of diagonal elements: all you have to do is take the inverse of the diagonal elements, for example:

$$\mathbf{D}^{-1}\mathbf{D} = \begin{bmatrix} -0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.25 \end{bmatrix} \begin{bmatrix} -2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{0.2}$$

With this decomposition (of **A** into **D** + **E**), we can write:

$$\begin{aligned} \mathbf{y} = \mathbf{Ax} &= (\mathbf{D} + \mathbf{E})\mathbf{x} \\ \mathbf{y} - \mathbf{Ex} &= \mathbf{Dx} \\ \mathbf{x} &= \mathbf{D}^{-1}\mathbf{y} - \mathbf{D}^{-1}\mathbf{Ex} \end{aligned} \tag{0.3}$$

and note $\mathbf{D}^{-1}\mathbf{y}$ and $\mathbf{D}^{-1}\mathbf{E}$ are both easy to calculate, and only have to be calculated once at the start of the iteration process. The Jacobi iteration starts with an initial guess at the value of **x**, written as **x**[0], and then iterates according to:

$$\mathbf{x}[n+1] = \mathbf{D}^{-1}\mathbf{y} - \mathbf{D}^{-1}\mathbf{Ex}[n] \tag{0.4}$$

and these iterations are easy to calculate, since **D** is easy to invert. For a wide-range of matrices **A**, this iteration converges to the answer. The problem is it doesn't always work, and it doesn't always converge very quickly.

1.1.1 Example of the Jacobi Method

Consider the set of simultaneous linear equations written in matrix form as:

$$\begin{bmatrix}1\\1\\1\end{bmatrix}=\begin{bmatrix}-2&1&-1\\-1&2&1\\2&-1&4\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}\tag{0.5}$$

Then start with a guess at **x**, perhaps **x** = [0; 0; 0]. This is the same matrix **A** as discussed above, so we have:

$$\mathbf{D}^{-1}=\begin{bmatrix}-0.5&0&0\\0&0.5&0\\0&0&0.25\end{bmatrix}\qquad\mathbf{E}=\begin{bmatrix}0&1&-1\\-1&0&1\\2&-1&0\end{bmatrix}\tag{0.6}$$

and therefore:

$$\mathbf{D}^{-1}\mathbf{E}=\begin{bmatrix}-0.5&0&0\\0&0.5&0\\0&0&0.25\end{bmatrix}\begin{bmatrix}0&1&-1\\-1&0&1\\2&-1&0\end{bmatrix}=\begin{bmatrix}0&-0.5&0.5\\-0.5&0&0.5\\0.5&-0.25&0\end{bmatrix}\tag{0.7}$$

and

$$\mathbf{D}^{-1}\mathbf{y}=\begin{bmatrix}-0.5&0&0\\0&0.5&0\\0&0&0.25\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}=\begin{bmatrix}-0.5\\0.5\\0.25\end{bmatrix}\tag{0.8}$$

and these can be pre-computed before the iteration starts. The first iteration then produces:

$$\mathbf{x}[1]=\begin{bmatrix}-0.5\\0.5\\0.25\end{bmatrix}-\begin{bmatrix}0&-0.5&0.5\\-0.5&0&0.5\\0.5&-0.25&0\end{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix}=\begin{bmatrix}-0.5\\0.5\\0.25\end{bmatrix}\tag{0.9}$$

the second iteration, then gives:

$$\mathbf{x}[2]=\begin{bmatrix}-0.5\\0.5\\0.25\end{bmatrix}-\begin{bmatrix}0&-0.5&0.5\\-0.5&0&0.5\\0.5&-0.25&0\end{bmatrix}\begin{bmatrix}-0.5\\0.5\\0.25\end{bmatrix}=\begin{bmatrix}-0.375\\0.125\\0.625\end{bmatrix}\tag{0.10}$$

the third gives:

$$\mathbf{x}[3]=\begin{bmatrix}-0.5\\0.5\\0.25\end{bmatrix}-\begin{bmatrix}0&-0.5&0.5\\-0.5&0&0.5\\0.5&-0.25&0\end{bmatrix}\begin{bmatrix}-0.375\\0.125\\-0.625\end{bmatrix}=\begin{bmatrix}-0.75\\0\\-0.4688\end{bmatrix}\tag{0.11}$$

and so on. The exact answer to this problem is:

$$\mathbf{x}_{opt} = \begin{bmatrix} -2 & 1 & -1 \\ -1 & 2 & 1 \\ 2 & -1 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1/3 \\ 2/3 \end{bmatrix} \tag{0.12}$$

and knowing this, it's possible to keep track of how well the algorithm is doing by calculating the square error $\|\mathbf{x}[n] - \mathbf{x}_{opt}\|^2$ at each step. For the first eight iterations, the results are:

	Vector $\mathbf{x}[\mathrm{n}]^T$ [1]	Square Error $\left\|\mathbf{x}[n]-\mathbf{x}_{opt}\right\|^2$
$\mathbf{x}[0]$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	1.5556
$\mathbf{x}[1]$	$\begin{bmatrix}-0.5 & 0.25 & 0.5625\end{bmatrix}^T$	1.1181
$\mathbf{x}[2]$	$\begin{bmatrix}-0.3750 & 0.1250 & 0.6250\end{bmatrix}^T$	0.6024
$\mathbf{x}[3]$	$\begin{bmatrix}-0.7500 & 0 & 0.4688\end{bmatrix}^T$	0.2128
$\mathbf{x}[4]$	$\begin{bmatrix}-0.7344 & -0.1094 & 0.6250\end{bmatrix}^T$	0.1225
$\mathbf{x}[5]$	$\begin{bmatrix}-0.8672 & -0.1797 & 0.5898\end{bmatrix}^T$	0.0471
$\mathbf{x}[6]$	$\begin{bmatrix}-0.8848 & -0.2285 & 0.6387\end{bmatrix}^T$	0.0250
$\mathbf{x}[7]$	$\begin{bmatrix}-0.9336 & -0.2617 & 0.6353\end{bmatrix}^T$	0.0105
$\mathbf{x}[8]$	$\begin{bmatrix}-0.9485 & -0.2844 & 0.6514\end{bmatrix}^T$	0.0053

(A reminder: the final answer should be $[-1 \quad -1/3 \quad 2/3]^T$.)

1.1.2 Convergence of the Jacobi Method

One problem with the Jacobi method is that it doesn't always work. To see why, and to understand how the method works at all, first express the result after n iterations $\mathbf{x}[n]$ as the sum of the real answer $\mathbf{x}_{opt}$, and an error term, $\mathbf{e}[n]$, so that:

$$\mathbf{x}[n] = \mathbf{x}_{opt} + \mathbf{e}[n] \tag{0.13}$$

Then, after the next iteration, the error term will be:

$$\begin{aligned} \mathbf{e}[n+1] &= \mathbf{x}[n+1] - \mathbf{x}_{opt} \\ &= \mathbf{D}^{-1}\mathbf{y} - \mathbf{D}^{-1}\mathbf{E}\mathbf{x}[n] - \mathbf{x}_{opt} \\ &= \mathbf{D}^{-1}\mathbf{y} - \mathbf{D}^{-1}\mathbf{E}\mathbf{x}_{opt} - \mathbf{D}^{-1}\mathbf{E}\mathbf{e}[n] - \mathbf{x}_{opt} \end{aligned} \tag{0.14}$$

but if $\mathbf{x}_{opt}$ is the real solution, then:

[1] Note I'm writing this as the transpose of a row vector since this format is easier to fit on the page. All vectors in this book, unless otherwise stated, are column vectors, so the transpose of a vector is a row vector.

$$\mathbf{x}_{opt} = \mathbf{D}^{-1}\mathbf{y} - \mathbf{D}^{-1}\mathbf{E}\mathbf{x}_{opt} \tag{0.15}$$

so:

$$\begin{aligned}\mathbf{e}[n+1] &= \mathbf{x}_{opt} - \mathbf{D}^{-1}\mathbf{E}\mathbf{e}[n] - \mathbf{x}_{opt} \\ &= -\mathbf{D}^{-1}\mathbf{E}\mathbf{e}[n]\end{aligned} \tag{0.16}$$

In other words, the effect of each iteration is to pre-multiply the error vector **e**[*n*] by the matrix $-\mathbf{D}^{-1}\mathbf{E}$. If the modulus of the largest eigenvalue of $-\mathbf{D}^{-1}\mathbf{E}$ is less than one, then this iteration will converge[2], and if all the eigenvectors are very much smaller than one, then the iteration will converge very rapidly.

However, if at least one eigenvalue of $\mathbf{D}^{-1}\mathbf{E}$ is equal to or greater than one, then this method will fail.

A simple way to tell whether this method will work for any given matrix **A**: if the modulus of the term on the main diagonal is greater than the sum of moduli of the other terms in the row, then the iteration will converge[3]. In maths, this criterion is:

$$\left|\mathbf{A}_{ii}\right| > \sum_{j\neq i}\left|\mathbf{A}_{ij}\right| \tag{0.17}$$

and a matrix that satisfies this criterion is known as *row diagonally dominant*. The matrix we used in the example above:

$$\mathbf{A} = \begin{bmatrix} -2 & 1 & -1 \\ -1 & 2 & 1 \\ 2 & -1 & 4 \end{bmatrix} \tag{0.18}$$

meets this criterion (the eigenvalues of $\mathbf{D}^{-1}\mathbf{E}$ in this case are –0.683, 0.5 and 0.183, all have magnitudes less than one).

Now, suppose you were given the set of equations:

$$\begin{aligned} 1 &= -2x_1 + x_2 - x_3 \\ 1 &= 2x_1 - x_2 + 4x_3 \\ 1 &= -x_1 + 2x_2 + x_3 \end{aligned} \tag{0.19}$$

and asked to solve them. Could you use Jacobi iteration? Well, at first glance you might think not, since in matrix notation this would give:

[2] Think of the error term as being composed of components parallel to the eigenvectors of $-\mathbf{D}^{-1}\mathbf{E}$. Then, each iteration results in multiplying each of these components by the corresponding eigenvalue. If all the eigenvalues are less than one, then this will result in a smaller error term, since all the components are now smaller than they were.

[3] This condition is sufficient, but not necessary. There are matrices that do not meet this criterion, but which still converge to a solution using the Jacobi method. The necessary condition is that the moduli of all the eigenvalues of $\mathbf{D}^{-1}\mathbf{E}$ are less than one.

$$\begin{bmatrix}1\\1\\1\end{bmatrix}=\begin{bmatrix}-2 & 1 & -1\\ 2 & -1 & 4\\ -1 & 2 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} \tag{0.20}$$

and this matrix is not row diagonally dominant: for example, the largest value in the second row is in the third column. And indeed, it doesn't work: the magnitude of the largest eigenvalue of this matrix is 7.765. However, at second glance, you might spot that all you have to do is swap the second and third equations over, to get:

$$\begin{aligned}1&=-2x_1+x_2-x_3\\1&=-x_1+2x_2+x_3\\1&=2x_1-x_2+4x_3\end{aligned} \tag{0.21}$$

and this is now row-diagonally dominant. Jacobi will work.

Unfortunately, this trick doesn't always work. For example, consider the equations:

$$\begin{bmatrix}1\\1\\1\end{bmatrix}=\begin{bmatrix}-2 & 1 & -1\\ -1 & 2 & 7\\ 2 & -1 & 4\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} \tag{0.22}$$

The eigenvectors of $\mathbf{D}^{-1}\mathbf{E}$ in this case are –0.8, 0.4 + 0.925j and 0.4 – 0.925j. The absolute value of the second two eigenvalues is greater than one (just), so the Jacobi iteration technique will fail to converge in this case. We'll need to use something more powerful.

1.2 The Gauss-Seidel Iterative Method

Slightly more complicated than the Jacobi method, but very much along the same lines, is the *Gauss-Seidel method*. The Gauss-Seidel method expresses the matrix **A** as the sum of a diagonal matrix, an upper triangular matrix, and a lower triangular matrix:

$$\mathbf{A}=\begin{bmatrix}-2 & 1 & -1\\ -1 & 2 & 1\\ 2 & -1 & 4\end{bmatrix}=\mathbf{D}-\mathbf{U}-\mathbf{L}=\begin{bmatrix}-2 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 4\end{bmatrix}-\begin{bmatrix}0 & -1 & 1\\ 0 & 0 & -1\\ 0 & 0 & 0\end{bmatrix}-\begin{bmatrix}0 & 0 & 0\\ 1 & 0 & 0\\ -2 & 1 & 0\end{bmatrix} \tag{0.23}$$

(Note the triangular matrices **U** and **L** are conventionally defined as the negative of the corresponding elements of **A**: the previous equation is the normal way of specifying this method.)

With this decomposition (of **A** into **D** – **U** – **L**), we can write:

$$\begin{aligned}\mathbf{y}=\mathbf{Ax}&=(\mathbf{D}-\mathbf{U}-\mathbf{L})\mathbf{x}\\ \mathbf{y}+\mathbf{Ux}&=\left(\mathbf{D}-\mathbf{L}\right)\mathbf{x}\\ \mathbf{x}&=\left(\mathbf{D}-\mathbf{L}\right)^{-1}\left(\mathbf{y}+\mathbf{Ux}\right)\end{aligned} \tag{0.24}$$

The Gauss-Seidel iteration starts with an initial guess of **x**[0] at the value of **x**, and then iterates according to:

$$\mathbf{x}[n+1] = (\mathbf{D}-\mathbf{L})^{-1}(\mathbf{y}+\mathbf{U}\mathbf{x}[n]) \tag{0.25}$$

This is a similar approach to the Jacobi method, except that now it appears we have to invert the matrix (**D** – **L**), which is not diagonal, rather than the diagonal matrix **D**. Surely that's much more difficult to do? So what's the advantage of this method? To answer that question, I'll have to talk about the Jacobi method in a bit more detail.

1.2.1 The Jacobi Method Step-by-Step

Consider what happens at each iteration step of the Jacobi method:

$$\mathbf{x}[n] = \mathbf{D}^{-1}\mathbf{y} - \mathbf{D}^{-1}\mathbf{E}\mathbf{x}[n] \tag{0.26}$$

To calculate the new vector **x**[n+1], each element must be worked out in turn. For the i^{th} element, this means calculating:

$$\mathbf{x}[n+1]_i = \left(\mathbf{D}^{-1}\mathbf{y}\right)_i - \sum_k \left(\mathbf{D}^{-1}\mathbf{E}\right)_{ik} \mathbf{x}[n]_k \tag{0.27}$$

which is actually a lot easier than it looks. Firstly, since **D** is a diagonal matrix, the inverse of **D** is also diagonal, and has elements that are the inverse of the elements in **D**. This leads to:

$$\mathbf{x}[n+1]_i = \frac{\mathbf{y}_i - \sum_k \mathbf{E}_{ik}\mathbf{x}[n]_k}{\mathbf{D}_{ii}} \tag{0.28}$$

and this can be further simplified by noting that **D** is the diagonal elements of **A**, and **E** is the non-diagonal elements, so $\mathbf{D}_{ii} = \mathbf{A}_{ii}$, and $\mathbf{E}_{ik} = \mathbf{A}_{ik}$ except for $\mathbf{E}_{ii}$, which is zero. So, we could write this in terms of the original matrix **A** as:

$$\mathbf{x}[n+1]_i = \frac{\mathbf{y}_i - \sum_{k\neq i} \mathbf{A}_{ik}\mathbf{x}[n]_k}{\mathbf{A}_{ii}} \tag{0.29}$$

Now, back to the Gauss-Seidel.

1.2.2 The Gauss-Seidel Method Step-by-Step

Iterations usually converge faster if the most recent values are used whenever possible. So, after calculating the new first element $\mathbf{x}[n+1]_1$, why not use this value immediately to calculate the next element $\mathbf{x}[n+1]_2$ rather than waiting for the whole of the vector **x**[n+1] to be calculated using only the terms in the previous vector **x**[n]?

That's exactly what the Gauss-Seidel method does. After calculating the first element $\mathbf{x}[n+1]_1$, we replace the value of $\mathbf{x}[n]_1$ with this new value, and carry on, using this new vector with a first element of $\mathbf{x}[n+1]_1$, and all the elements those of **x**[n]. Then, we use this vector to calculate the second element $\mathbf{x}[n+1]_2$. (The result won't be the same as if the Jacobi method was being used, since we're no longer starting with the vector **x**[n], we're using this strange new hybrid vector.) That'll give us a new second element. Replace the second element with this new value, and use this new vector to calculate the third element in $\mathbf{x}[n+1]$. Repeat for all the elements, and for the next iteration go back to the start and start over.

In effect, we're doing:

$$\mathbf{x}[n+1]_i = \frac{\mathbf{y}_i - \sum_{k<i} \mathbf{E}_{ik}\mathbf{x}[n+1]_k - \sum_{k\geq i} \mathbf{E}_{ik}\mathbf{x}[n]_k}{\mathbf{D}_{ii}} \tag{0.30}$$

at every step, where **E** is a matrix of the non-diagonal elements in **A**, and **D** is the matrix of just the diagonal elements, just like before. Since $\mathbf{E}_{ii}$ is zero, we don't need to consider the term in the second summation when $k = i$, and we could just write this as:

$$\mathbf{x}[n+1]_i = \frac{\mathbf{y}_i - \sum_{k<i} \mathbf{A}_{ik}\mathbf{x}[n+1]_k - \sum_{k>i} \mathbf{A}_{ik}\mathbf{x}[n]_k}{\mathbf{A}_{ii}} \tag{0.31}$$

This is the only difference between the Gauss-Seidel algorithm and the Jacobi algorithm: with the Gauss-Seidel algorithm, the new values of $\mathbf{x}[n+1]$ are used as soon as they become available, you don't wait for all of the vector $\mathbf{x}[n+1]$ to be calculated before starting to use any of the terms.

If you're going to derive the form of the Gauss-Seidel in terms of triangular and diagonal matrices, then it's much easier to start with equation (0.31) and note that the terms of $\mathbf{A}_{ik}$ in which $k < i$ are just all the terms in the lower triangular matrix, the one we've called $-\mathbf{L}$. Likewise, the terms of $\mathbf{A}_{ik}$ in which $k > i$ are the terms in the upper triangular matrix, the one we've called $-\mathbf{U}$. So:

$$\mathbf{x}[n+1]_i = \frac{\mathbf{y}_i + \sum_{k} \mathbf{L}_{ik}\mathbf{x}[n+1]_k + \sum_{k} \mathbf{U}_{ik}\mathbf{x}[n]_k}{\mathbf{D}_{ii}} \tag{0.32}$$

and since we're now summing over the full range of k in both cases, we can write this in matrix form:

$$\mathbf{D}\mathbf{x}[n+1] = \mathbf{y} + \mathbf{L}\mathbf{x}[n+1] + \mathbf{U}\mathbf{x}[n] \tag{0.33}$$

which with a bit of algebraic manipulation gives:

$$\mathbf{x}[n+1] = (\mathbf{D}-\mathbf{L})^{-1}(\mathbf{y} + \mathbf{U}\mathbf{x}[n]) \tag{0.34}$$

which is identical to equation (0.25). Although it looks more complicated, in fact it doesn't require any more calculations than the Jacobi method.

1.2.3 Example of the Gauss-Seidel Method

Doing the same example as done in section 1.1.1 results in the following series of approximations, and square errors. Note that the Gauss-Seidel in this case converges much faster than the Jacobi, due to the use of more recent information in the algorithm.

	Vector $\mathbf{x}$[n]	Square Error $\lVert\mathbf{x}[n]-\mathbf{x}\rVert^2$

$\mathbf{x}[0]$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$	1.5556
$\mathbf{x}[1]$	$\begin{bmatrix}-0.5 & 0.25 & 0.5625\end{bmatrix}^T$	0.6011
$\mathbf{x}[2]$	$\begin{bmatrix}-0.6563 & -0.1094 & 0.5508\end{bmatrix}^T$	0.1818
$\mathbf{x}[3]$	$\begin{bmatrix}-0.8301 & -0.1904 & 0.6174\end{bmatrix}^T$	0.0517
$\mathbf{x}[4]$	$\begin{bmatrix}-0.9039 & -0.2607 & 0.6368\end{bmatrix}^T$	0.0154
$\mathbf{x}[5]$	$\begin{bmatrix}-0.9487 & -0.2928 & 0.6512\end{bmatrix}^T$	0.0045
$\mathbf{x}[6]$	$\begin{bmatrix}-0.9720 & -0.3116 & 0.6581\end{bmatrix}^T$	0.0013
$\mathbf{x}[7]$	$\begin{bmatrix}-0.9848 & -0.3215 & 0.6621\end{bmatrix}^T$	0.0004
$\mathbf{x}[8]$	$\begin{bmatrix}-0.9918 & -0.3269 & 0.6642\end{bmatrix}^T$	0.0001

1.2.4 Convergence of the Gauss-Seidel Method

To work out when the Gauss-Seidel method converges, we can proceed in the same way as for the simpler Jacobi case above. First, express **x**[n] as the sum of the actual answer $\mathbf{x}_{opt}$, and an error term, **e**[n], so that:

$$\mathbf{x}[n] = \mathbf{x}_{opt} + \mathbf{e}[n] \tag{0.35}$$

Then, after the next iteration, the error term will be:

$$\begin{aligned}\mathbf{e}[n+1] = \mathbf{x}[n+1] - \mathbf{x}_{opt} &= (\mathbf{D}-\mathbf{L})^{-1}\mathbf{y} + (\mathbf{D}-\mathbf{L})^{-1}\mathbf{U}\mathbf{x}[n] - \mathbf{x}_{opt} \\ &= (\mathbf{D}-\mathbf{L})^{-1}\mathbf{y} + (\mathbf{D}-\mathbf{L})^{-1}\mathbf{U}\left(\mathbf{x}_{opt} + \mathbf{e}[n]\right) - \mathbf{x}_{opt}\end{aligned} \tag{0.36}$$

but if $\mathbf{x}_{opt}$ is the real solution, then:

$$\mathbf{x}_{opt} = (\mathbf{D}-\mathbf{L})^{-1}\mathbf{y} + (\mathbf{D}-\mathbf{L})^{-1}\mathbf{U}\mathbf{x}_{opt} \tag{0.37}$$

so:

$$\begin{aligned}\mathbf{e}[n+1] &= \mathbf{x}_{opt} + (\mathbf{D}-\mathbf{L})^{-1}\mathbf{U}\mathbf{e}[n] - \mathbf{x}_{opt} \\ &= (\mathbf{D}-\mathbf{L})^{-1}\mathbf{U}\mathbf{e}[n]\end{aligned} \tag{0.38}$$

So in this case, the error term is pre-multiplied by the matrix $(\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}$ after each iteration. This implies that if the largest eigenvalue of $(\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}$ is less than one, then this iteration will converge, and again, if all the eigenvectors are very much smaller than one, then the iteration will converge very rapidly. However, if at least one eigenvector is equal to or greater than one, then this method will fail.

Again, a sufficient (but not necessary) condition for convergence is that the matrix **A** is row diagonally dominant.

1.2.5 Comparison of the Gauss-Seidel and Jacobi Methods

The Gauss-Seidel is the more common and usually favoured method, for a number of reasons.

Firstly, and most importantly, it tends to converge faster. This is a result of using the new information (the updated values of the elements of the next estimate of **x**) as soon as they become available, and not waiting for all the elements in the vector to be updated before using any of them. Secondly, it doesn't require as much storage space: the elements of the vector **x** can be replaced one-by-one. Thirdly, there are some matrices for which Gauss-Seidel converges, but Jacobi does not.

(On the other hand, there are some matrices that Jacobi does converge for, and Gauss-Seidel doesn't, but there are fewer of these. You've got a better chance with the Gauss-Seidel.)

1.3 Preconditioning

Both the Jacobi and the Gauss-Seidel methods only converge when the eigenvectors of certain matrices ($\mathbf{D}^{-1}\mathbf{E}$ in the case of Jacobi and $(\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}$ in the case of Gauss-Seidel) all have magnitudes less than one. What if this is not true? And how do you know, before you start the iterations, whether it is true or not? In general, finding the eigenvalues of matrices is not a trivial task.

One solution is to first pre-multiply the equations by some other matrix **B**, so we're actually trying to solve the equations:

$$\mathbf{By} = \mathbf{BAx} \tag{0.39}$$

If we know **B**, then calculating **By** is easy, and this leaves us with a similar problem to before, except that we've now replaced the matrix **A** with **BA**. If we choose **B** so that **BA** has the required properties for convergence, then we can ensure that the iteration is successful. Unfortunately, finding a suitable matrix **B** to use is not always easy.

One possibility is using $\mathbf{B} = \mathbf{A}^H$. Then, we're trying to solve the equations:

$$\mathbf{A}^H\mathbf{y} = \mathbf{A}^H\mathbf{Ax} \tag{0.40}$$

$\mathbf{A}^H\mathbf{A}$ is a positive definite matrix, which among other things means that the Gauss-Seidel iteration method is guaranteed to converge[4] (although it might not converge very quickly).

1.4 Over-Relaxation Methods

When the eigenvalues of the matrix that multiplies the error term at each iteration are all positive, then these methods will converge to the correct solution in a series of steps, getting closer to the solution at each step. That means that the correct solution is always slightly further away than the next step. So: why not jump a little further each time? Instead of an iteration like:

$$\mathbf{x}[n+1] = (\mathbf{D} - \mathbf{L})^{-1}(\mathbf{y} + \mathbf{Ux}[n]) \tag{0.41}$$

[4] The proof of this is beyond the scope of this chapter. See the chapter on Matrix Properties under positive definite matrices for more information about this.

why not use:

$$\mathbf{x}\left[n+1\right]=w\left(\left(\mathbf{D}-\mathbf{L}\right)^{-1}\left(\mathbf{y}+\mathbf{U}\mathbf{x}[n]\right)-\mathbf{x}[n]\right)+\mathbf{x}[n] \tag{0.42}$$

where the parameter w is greater than one? Then you're moving further than you would with the Gauss-Seidel, and that might get you closer to the correct solution with each iteration? This is in general called over-relaxation, and in this particular case, the successive over-relaxation method. It can increase the speed of the convergence, at the expense of potentially making the iterations unstable, and causing divergence.

Actually, equation (0.42) is not the form most often used, because it's harder to calculate: it requires that all the elements of the vector **x**[n] are available when trying to calculate the vector **x**[n + 1], and one of the main advantages of the Gauss-Seidel method over the Jacobi is that it doesn't have to keep the last iteration, it can replace terms one-by-one. It's more usual to go back to the iteration form:

$$\mathbf{x}\left[n+1\right]_i=\frac{\mathbf{y}_i+\sum_k \mathbf{L}_{ik}\mathbf{x}[n+1]_k+\sum_k \mathbf{U}_{ik}\mathbf{x}[n]_k}{\mathbf{D}_{ii}} \tag{0.43}$$

which does not require the entire vector **x**[n] to be available to calculate **x**[n+1], and multiply the difference between each term **x**[n+1]$_i$ and **x**[n]$_i$ as they are calculated by w:

$$\mathbf{x}\left[n+1\right]_i=w\left(\frac{\mathbf{y}_i+\sum_k \mathbf{L}_{ik}\mathbf{x}[n+1]_k+\sum_k \mathbf{U}_{ik}\mathbf{x}[n]_k}{\mathbf{D}_{ii}}-\mathbf{x}[n]_i\right)+\mathbf{x}[n]_i \tag{0.44}$$

which with a bit of algebraic manipulation, gives:

$$\mathbf{x}\left[n+1\right]=\left(\mathbf{D}-w\mathbf{L}\right)^{-1}\left(w\mathbf{y}+w\mathbf{U}\mathbf{x}[n]+\left(1-w\right)\mathbf{D}\mathbf{x}[n]\right) \tag{0.45}$$

How to choose the best possible value of w is again beyond the scope of this chapter, however it's worth noting that values outside the range $0 < w < 2$ do not converge, and using $w = 1$ is just equivalent to using the Gauss-Seidel method.

1.5 Tutorial Questions

1) Try and solve the following systems of equations using the Jacobi and Gauss-Seidel iteration techniques (MATLAB or some other mathematical program would be of great use for this question, but you could just calculate the exact solution, and the result from the first iteration step, and see if the error term gets smaller). Which of them converge?

$$\text{a)}\ \begin{bmatrix}1\\1\\1\end{bmatrix}=\begin{bmatrix}2&3&5\\2&-2&1\\-2&1&-4\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}\qquad \text{b)}\ \begin{bmatrix}1\\1\\1\end{bmatrix}=\begin{bmatrix}5&1&2\\0&4&9\\6&1&4\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}$$

c) $\begin{bmatrix}1\\1\\1\end{bmatrix}=\begin{bmatrix}-3 & 7 & 4\\2 & -4 & -1\\3 & 4 & 2\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}$ d) $\begin{bmatrix}1\\1\\1\end{bmatrix}=\begin{bmatrix}-2 & 1 & -1\\-1 & 0 & 1\\2 & -1 & 4\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}$

2) In cases where the Jacobi and Gauss-Seidel approaches both converge, does the Gauss Seidel always converge faster?

3) Prove that the over-relaxation form of the Gauss-Seidel equation:

$$\mathbf{x}\left[n+1\right]_i = w\left(\frac{\mathbf{y}_i + \sum_k \mathbf{L}_{ik}\mathbf{x}[n+1]_k + \sum_k \mathbf{U}_{ik}\mathbf{x}[n]_k}{\mathbf{D}_{ii}} - \mathbf{x}[n]_i\right) + \mathbf{x}[n]_i$$

can be simplified to:

$$\mathbf{x}\left[n+1\right] = \left(\mathbf{D} - w\mathbf{L}\right)^{-1}\left(w\mathbf{y} + w\mathbf{U}\mathbf{x}[n] + \left(1-w\right)\mathbf{D}\mathbf{x}[n]\right)$$

4) The over-relaxation form in equation (0.45) was derived from the Gauss-Seidel method. Can you use the same over-relaxation idea, only starting with the Jacobi method? If so, derive the equivalent equation for this case.

5) The Gauss-Seidel method is guaranteed to converge if the matrix **A** is positive definite. What about the Jacobi method? Does this always converge is **A** is positive definite? (Either a proof or an example of a positive definite **A** that does not converge using the Jacobi method will be fine.)

6) For over-relaxation methods, when might a value of w of less than one be a good idea? How could you calculate the optimum value to use, and is it worth calculating?