Numer. Math. 32, 105 – 108 (1979)

Numerische Mathematik

Alternating Methods for Sets of Linear Equations

V. Conrad and Y. Wallach

Faculty of Electrical Engineering, Technion – Israel Institute of Technology, Haifa, Israel

Summary. A generalization of alternating methods for sets of linear equations is described and the number of operations calculated. It is shown that the lowest number of arithmetic operations is achieved in the SSOR algorithm.

Subject Classifications. AMS (MOS): 65F10; CR: 5.14.

Introduction

The authors have published two papers, [1, 2] which describe how the set

$$A x = b \tag{1}$$

may be solved with alternating, iterative algorithms. In [1], Gauss-Seidel and Jacobi "sweeps" alternate, in [2, 3] the Symmetric Successive Overrelaxation is used. In the first 25 %, in the second 50 % of operations may be saved. These computational methods are special cases of a general scheme, whereby any two explicit iterations may be combined in an alternating fashion. The present paper deals with the generalization.

Let

$$A = B_1 - C_1 = B_2 - C_2 \tag{2}$$

be two explicit splittings[1] of the matrix A associated with the iterative methods

$$B_1 \underline{x}^{(m+1)} = C_1 \underline{x}^{(m)} + \underline{b}, \tag{3}$$

$$B_2 \underline{x}^{(m+1)} = C_2 \underline{x}^{(m)} + \underline{b}. \tag{4}$$

Suppose the two methods are to be applied in an alternating way, i.e., $\underline{x}^{(1)}$ is obtained from $\underline{x}^{(0)}$ using (3), and $\underline{x}^{(2)}$ from $\underline{x}^{(1)}$ through (4), $\underline{x}^{(3)}$ from $\underline{x}^{(2)}$

[1] By an explicit splitting we mean that Eqs. (3) or (4) may be solved for $\underline{x}^{(m+1)}$ through back-substitution only (i.e. requiring no matrix inversion). This in effect means that non-zero elements of B_1 and B_2 lie on one side of the diagonal

0029-599X/79/0032/0105/$01.00

through (3), etc. The equations describing the alternating method are:

$$B_1 \underline{x}^{(m+1/2)} = C_1 \underline{x}^{(m)} + \underline{b}, \tag{5a}$$

$$B_2 \underline{x}^{(m+1)} = C_2 \underline{x}^{(m+1/2)} + b, \quad m \geqq 0. \tag{5b}$$

We define now the following sets of pairs of indices:

$$\begin{aligned} \mathscr{B}_1 &= \{(i,j) | (B_1)_{ij} \neq 0,\ i \neq j\} \\ \mathscr{B}_2 &= \{(i,j) | (B_2)_{ij} \neq 0,\ i \neq j\} \\ \mathscr{C}_1 &= \{(i,j) | (C_1)_{ij} \neq 0,\ i \neq j\} \\ \mathscr{C}_2 &= \{(i,j) | (C_2)_{ij} \neq 0,\ i \neq j\}. \end{aligned} \tag{6}$$

As is the case with most iterative methods, we consider the splittings to correspond to a decomposition of the matrix A, i.e.

$$\mathscr{B}_1 \cup C_1 = S, \quad \mathscr{B}_1 \cap C_1 = \phi, \tag{7}$$

$$\mathscr{B}_2 \cup C_2 = S, \quad \mathscr{B}_2 \cap C_2 = \phi \tag{8}$$

where

$$\mathscr{S} = \{(i,j) | (A)_{ij} \neq 0,\ i \neq j\}. \tag{9}$$

Now denote

$$\mathscr{B}_2 \cap \mathscr{C}_1 = \mathscr{N}, \tag{10}$$

$$\mathscr{B}_1 \cap \mathscr{C}_2 = \mathscr{M}. \tag{11}$$

Theorem. Given the storage of an additional n-vector, it is possible to perform the alternating iterative method of (5) in no more than

$$\mu = 2|\mathscr{S}| - |\mathscr{M}| - |\mathscr{N}| \tag{12}$$

m/d (multiplications or divisions) per iteration.

Constructive Proof

We outline an algorithm which achieves this bound. It is assumed that we have at our disposal an auxiliary n-vector $\underline{y}$. We also assume that initially $\underline{y}$ holds $C_1 \underline{x}^{(m)}$, and that $a_{ii} = 1$, $i = 1, 2, \ldots, n$.

(a) The iteration starts by computing $\underline{x}^{(m+1/2)}$ through (5a). Since the splitting is explicit, this amounts to back-substitution. While computing $\underline{x}^{(m+1/2)}$, we form the sum $\sum_{(i,j) \in \mathscr{M}} a_{ij} x_j^{(m+1/2)}$ and store it in y_i. This step requires $|\mathscr{B}_1|$ multiplications.

(b) Next, $\underline{x}^{(m+1)}$ is evaluated through (5b). We note that some of the operations required in computing $C_2 \underline{x}^{(m+1/2)}$ have already been performed in step (a) and stored in $\underline{y}$. In fact,

$$(C_2 \underline{x}^{(m+1/2)})_k = \sum_{(k,j)\in\mathscr{C}_2} a_{kj} x_j^{(m+1/2)} = \Big(\sum_{(k,j)\in\mathscr{M}} + \sum_{(k,j)\in\mathscr{C}_2-\mathscr{M}}\Big) a_{kj} x_j^{(m+1/2)} \tag{13}$$

and the first sum is stored in y_k and neex not be computed again. Since by (11) $\mathscr{M}\subset\mathscr{C}_2$, it follows that $|\mathscr{C}_2-\mathscr{M}|=|\mathscr{C}_2|-|\mathscr{M}|$. Therefore the second summation calls for $|\mathscr{C}_2|-|\mathscr{M}|$ multiplications. After computing $(C_2 \underline{x}^{(m+1/2)})_k$, y_k is not needed and may be over-written. In analogy with step (a), we compute now $\underline{x}^{(m+1)}$, the back-substitution requiring $|\mathscr{B}_2|$ multiplications of the form $\sum_{(i,j)\in\mathscr{B}_2} a_{ij} x^{(m+1)}$. For each k, the partial sum $\sum_{(k,j)\in\mathscr{N}} a_{ij} x^{(m+1)}$ is computed separately and saved in y_k.

(c) Before reverting to (a) to start a new iteration, the auxiliary vector y_k must contain $C_1 \underline{x}^{(m+1)}$, i.e.

$$y_k = \sum_{(k,j)\in\mathscr{C}_1} a_{ij} \underline{x}_j^{(m+1)} = \Big(\sum_{(k,j)\in\mathscr{N}} + \sum_{(k,j)\in\mathscr{C}_1-\mathscr{N}}\Big) a_{ij} \underline{x}_j^{(m+1)} \tag{14}$$

must hold. Only the second sum must be computed (the first is already stored in y_k). This calls for $|\mathscr{C}_1|-|\mathscr{N}|$ multiplications. Altogether there are

$$|\mathscr{B}_1|+|\mathscr{C}_2|-|\mathscr{M}|+|\mathscr{B}_2|+|\mathscr{C}_1|-|\mathscr{N}| = 2|\mathscr{S}|-|\mathscr{M}|-|\mathscr{N}|$$

multiplications. Q.E.D.

Remark 1. Over or underrelaxation may be easily incorporated into the above scheme. After computing $x_j^{(m+1/2)}$ from (5a), one replaces it by $x_j^{(m+1/2)} + \omega(x_j^{(m+1/2)} - x_j^{(m)})$, where ω is the relaxation factor. A similar formula applies to the second half-step.

A single iteration of a given iterative method requires $|\mathscr{S}|m/d$. Hence if the two half-steps of the alternating method (5) are programmed in the conventional way, i.e. as if they were completely independent iterations, the resulting program calls for $2|\mathscr{S}|m/d$. It may also be shown that this algorithm leads to similar savings in the number of additions and subtractions.

Since the loop structure implied by the algorithm is similar to that of conventional programming (with the exception of storing the y_k's) no increase in index calculation and loop control overhead is anticipated.

We shall now show how various algorithms are derived from this general scheme. Suppose first that $\mathscr{B}_1=\mathscr{B}_2$, hence $\mathscr{C}_1=\mathscr{C}_2$; by (10), (11), $\mathscr{M}=\mathscr{N}=\phi$ and the number of multiplications is $2|\mathscr{S}|$; therefore if two half-steps are identical, there is no gain. (This shows that stationary processes, such as the Gauss-Seidel or Jacobi algorithms, do not benefit from the above result.)

Consider now the case

$$B_1 = L + D, \qquad C_1 = -U, \tag{15}$$

$$B_2 = D, \qquad C_2 = -(L+U) \tag{16}$$

which corresponds to the "Seidel-Jacobi" SJ-method of [1]. Putting

$$\mathscr{L} = \{(ij) | (L)_{ij} \neq 0,\ i \neq j\} \tag{17}$$

and similarly

$$\mathcal{U}=\{(ij)|(U)_{ij}\neq 0,\ i\neq j\} \tag{18}$$

we see that in this case

$$\mathcal{N}=\phi, \quad \mathcal{M}=\mathcal{L} \tag{19}$$

and the composite iteration requires $2|\mathcal{S}|-|\mathcal{L}|$ multiplications. In the case of a full matrix, $|\mathcal{S}|=n^2$, $|\mathcal{L}|\simeq n^2/2$ hence $\sim 3n^2/2$ multiplications are required – a saving of 25 %.

Finally, if

$$B_1=D+L, \quad C_1=-U, \tag{20}$$

$$B_2=D+U, \quad C_2=-L \tag{21}$$

which corresponds to back-and-forth Seidel (or SSOR, if overrelaxation is introduced) we see that

$$\mathcal{M}=\mathcal{U}, \quad \mathcal{N}=\mathcal{L}, \tag{22}$$

$$\begin{aligned}\mu&=|\mathcal{B}_1|+|\mathcal{C}_1|+|\mathcal{B}_2|+|\mathcal{C}_2|-|\mathcal{M}|-|\mathcal{N}|\\&=2(|\mathcal{L}|+|\mathcal{U}|)-|\mathcal{U}|-|\mathcal{L}|=|\mathcal{L}|+|\mathcal{U}|=|\mathcal{S}|\end{aligned} \tag{23}$$

which is a gain of 50 % over the straightforward algorithm for the case of a full matrix.

Remark 2. The gain achieved in the last example is the maximum that can be obtained through the above scheme. Since $\mathcal{M}\subset\mathcal{B}_1$ and $\mathcal{N}\subset\mathcal{C}_1$ by (10) and (11), and also $\mathcal{B}_1\cap\mathcal{C}_1=\phi$ by (7), it follows that

$$|\mathcal{M}|+|\mathcal{N}|\leqq|\mathcal{B}_1\cup\mathcal{C}_1|=|\mathcal{S}|.$$

Substituting this in (12) we get

$$\mu\geqq 2|\mathcal{S}|-|\mathcal{S}|=|\mathcal{S}|. \tag{24}$$

We see that SSOR attains this optimum gain.

Remark 3. Computational savings gained by alternating two different, iterative schemes, extend to the case of non-explicit splittings. Since, in these cases, inversion is required, the operation count is more complicated.

References

1. Conrad, V., Wallach, Y.: Iterative solution of linear equations on a parallel processor system. Trans. IEEE (Computers), **C-26**, 838 – 847 (1977)
2. Conrad, V., Wallach, Y.: A faster SSOR Algorythm. Numer. Math. **27**, 371 – 372 (1977)
3. Niethammer, W.: Relaxation bei komplexen Matrizen. Math. Zeitschr. **86**, 34 – 40 (1964)

Received December 19, 1977