Block Matrices and Algorithms

P. Sam Johnson

National Institute of Technology Karnataka (NITK)
Surathkal, Mangalore, India

Introduction

Having a facility with block matrix notation is crucial in matrix computations because it simplifies the derivation of many central algorithms.

Moreover, “block algorithms” are increasingly important in high performance computing. By a block algorithm we essentially mean an algorithm that is rich in matrix-matrix multiplication.

Algorithms of this type turn out to be more efficient in many computing environments than those that are organized at a lower linear algebraic level.

Block Matrix Notation

Column and row partitionings are special cases of matrix blocking. In general we can partition both the rows and columns of an m-by-n matrix A to obtain

$$A = \begin{array}{cc} \begin{bmatrix} A_{11} & \cdots & A_{1r} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qr} \end{bmatrix} & \begin{array}{c} m_1 \\ \\ m_q \end{array} \\ \begin{array}{ccc} n_1 & \qquad & n_r \end{array} & \end{array}$$

where $m_1 + \cdots + m_q = m_1$ $n_1 + \cdots + n_r = n_1$ and $A_{\alpha\beta}$ designates the (α, β) block or submatrix. With this notation, block $A_{\alpha\beta}$ has dimension m_α-by-n_β and we say that $A = (A_{\alpha\beta})$ is a q-by-r block matrix.

Block Matrix Manipulation

Block matrices combine just like matrices with scalar entries as long as certain dimension requirements are met. For example, if

$$B = \begin{array}{cc} \left[\begin{array}{ccc} B_{11} & \cdots & B_{1r} \\ \vdots & & \vdots \\ B_{q1} & \cdots & B_{qr} \end{array}\right] & \begin{array}{c} m_1 \\ \\ m_q \end{array} \\ \begin{array}{ccc} n_1 & \quad\quad & n_r \end{array} & \end{array},$$

then we say that B is partitioned conformably with the matrix A above. The sum $C = A + B$ can also be regarded as a q-by-r block matrix:

$$C = \begin{bmatrix} C_{11} & \cdots & C_{1r} \\ \vdots & & \vdots \\ C_{q1} & \cdots & C_{qr} \end{bmatrix} = \begin{bmatrix} A_{11} + B_{11} & \cdots & A_{1r} + B_{1r} \\ \vdots & & \vdots \\ A_{q1} + B_{q1} & \cdots & A_{qr} + B_{qr} \end{bmatrix}.$$

Block Matrix Manipulation (Contd...)

The multiplication of block matrices is a little trickier. We start with a pair of lemmas.

Lemma 1.

If $A \in \mathbb{R}^{m\times p}$, $B \in \mathbb{R}^{p\times n}$,

$$A = \begin{array}{cc} \begin{bmatrix} A_1 \\ \vdots \\ A_q \end{bmatrix} & \begin{array}{c} m_1 \\ \\ m_q \end{array} \end{array} \qquad B = \begin{array}{ccc} [B_1 & ,\ldots, & B_r] \\ n_1 & & n_r \end{array},$$

then

$$AB = C = \begin{array}{cc} \begin{bmatrix} C_{11} & \cdots & C_{1r} \\ \vdots & & \vdots \\ C_{q1} & \cdots & C_{qr} \end{bmatrix} & \begin{array}{c} m_1 \\ \\ m_q \end{array} \\ \begin{array}{ccc} n_1 & \qquad\qquad & n_r \end{array} & \end{array}$$

where $C_{\alpha\beta} = A_\alpha B_\beta$ *for* $\alpha = 1:q$ *and* $\beta = 1:r$.

Block Matrix Manipulation (Contd...)

Proof. First we relate scalar entries in block $C_{\alpha\beta}$ to scalar entries in C. For $1 \leq \alpha \leq q$, $1 \leq \beta \leq r$, $1 \leq i \leq m_\alpha$, and $1 \leq j \leq n_\beta$ we have

$$[C_{\alpha\beta}]_{ij} = c_{\lambda+i,\mu+j}$$

where

$$\begin{aligned} \lambda &= m_1 + \cdots + m_{\alpha-1} \\ \mu &= n_1 + \cdots + n_{\beta-1}. \end{aligned}$$

But

$$c_{\lambda+i,\mu+j} = \sum_{k=1}^{p} a_{\lambda+i,k} b_{k,\mu+j} = \sum_{k=1}^{p} [A_\alpha]_{ik} [B_\beta]_{kj} = [A_\alpha B_\beta]_{ij}.$$

Thus, $C_{\alpha\beta} = A_\alpha B_\beta$.

Block Matrix Manipulation (Contd...)

Lemma 2.

If $A \in \mathbb{R}^{m\times p}$, $B \in \mathbb{R}^{p\times n}$,

$$A = \begin{array}{cccc} [A_1 & ,\cdots, & A_s] \\ p_1 & & p_s \end{array}, \quad \text{and} \quad B = \begin{array}{c} \left[\begin{array}{c} B_1 \\ \vdots \\ B_s \end{array}\right] \end{array} \begin{array}{c} p_1 \\ \\ p_s \end{array},$$

then

$$AB = C = \sum_{\gamma=1}^{s} A_\gamma B_\gamma .$$

Block Matrix Manipulation (Contd...)

Proof. We set $s = 2$ and leave the general s case to the reader. For $1 \leq i \leq m$ and $1 \leq j \leq n$ we have

$$\begin{aligned} c_{ij} &= \sum_{k=1}^{p} a_{ik} b_{kj} = \sum_{k=1}^{p_1} a_{ik} b_{kj} + \sum_{k=p_1+1}^{p_1+p_2} a_{ik} b_{kj} \\ &= [A_1 B_1]_{ij} + [A_2 B_2]_{ij} = [A_1 B_1 + A_2 B_2]_{ij}. \end{aligned}$$

Thus, $C = A_1 B_1 + A_2 B_2$.

For general block matrix multiplication we have the following result:

Block Matrix Manipulation (Contd...)

Theorem 3.

If

$$A = \begin{array}{cc} \begin{bmatrix} A_{11} & \cdots & A_{1s} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qs} \end{bmatrix} & \begin{array}{c} m_1 \\ \\ m_q \end{array} \\ \begin{array}{ccc} p_1 & \qquad & p_s \end{array} & \end{array}, \quad B = \begin{array}{cc} \begin{bmatrix} B_{11} & \cdots & B_{1r} \\ \vdots & & \vdots \\ B_{a1} & \cdots & B_{ar} \end{bmatrix} & \begin{array}{c} p_1 \\ \\ p_s \end{array} \\ \begin{array}{ccc} n_1 & \qquad & n_r \end{array} & \end{array},$$

and we partition the product $C = AB$ as follows,

$$C = \begin{array}{cc} \begin{bmatrix} C_{11} & \cdots & C_{1r} \\ \vdots & & \vdots \\ C_{q1} & \cdots & C_{qr} \end{bmatrix} & \begin{array}{c} m_1 \\ \\ m_q \end{array} \\ \begin{array}{ccc} n_1 & \qquad & n_r \end{array} & \end{array},$$

then

$$C_{\alpha\beta} = \sum_{\gamma=1}^{s} A_{\alpha\gamma} B_{\gamma\beta} \quad \alpha = 1 : q, \quad \beta = 1 : r.$$

Block Matrix Manipulation (Contd...)

A very important special case arises if we set $s = 2$, $r = 1$, and $n_1 = 1$:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 \\ A_{21}x_1 + A_{22}x_2 \end{bmatrix}.$$

This partitioned matrix-vector product is used over and over again in subsequent chapters.

Submatrix Designation

As with "ordinary" matrix multiplication, block matrix multiplication can be organized in several ways. To specify the computations precisely, we need some notation.

Suppose $A \in \mathbb{R}^{m\times n}$ and that $i = (i_1, \dots, i_r)$ and $j = (j_1, \dots, j_c)$ are integer vectors with the property that

$$\begin{gathered} i_1, \dots, i_r \in \{1, 2, \dots, m\} \\ j_1, \dots, j_c \in \{1, 2, \dots, n\}. \end{gathered}$$

We let $A(i, j)$ denote the r-by-c submatrix

$$A(i,j) = \begin{bmatrix} A(i_1, j_1) & \cdots & A(i_1, j_c) \\ \vdots & & \vdots \\ A(i_r, j_1) & \cdots & A(i_r, j_c) \end{bmatrix}.$$

Submatrix Designation (Contd...)

If the entries in the subscript vectors i and j are contiguous, then the "colon" notation can be used to define $A(i, j)$ in terms of the scalar entries in A.

In particular, if $1 \leq i_1 \leq i_2 \leq m$ and $1 \leq j_1 \leq j_2 \leq n$, then $A(i_1 : i_2, j_1 : j_2)$ is the submatrix obtained by extracting rows i_1 through i_2 and columns j_1 through j_2, e.g,

$$A(3:5, 1:2) = \begin{bmatrix} a_{31} & a_{32} \\ a_{41} & a_{42} \\ a_{51} & a_{52} \end{bmatrix}.$$

While on the subject of submatrices, recall from §1.1.8 that if i and j are scalars, then $A(i, :)$ designates the ith row of A and $A(:, j)$ designates the jth column of A.

Block Matrix Times Vector

An important situation covered by Theorem 3 is the case of a block matrix times vector. Let us consider the details of the gaxpy $y = Ax + y$ where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, and

$$A = \begin{array}{c} \begin{bmatrix} A_1 \\ \vdots \\ A_q \end{bmatrix} \end{array} \begin{array}{c} m_1 \\ \\ m_q \end{array} \qquad y = \begin{array}{c} \begin{bmatrix} y_1 \\ \vdots \\ y_q \end{bmatrix} \end{array} \begin{array}{c} m_1 \\ \\ m_q \end{array} .$$

We refer to A_i as the ith block row. If $m.vec = (m_1, \ldots, m_q)$ is the vector of block row "heights", then from

$$\begin{bmatrix} y_1 \\ \vdots \\ y_q \end{bmatrix} = \begin{bmatrix} A_1 \\ \vdots \\ A_q \end{bmatrix} x + \begin{bmatrix} y_1 \\ \vdots \\ y_q \end{bmatrix}.$$

Block Matrix Times Vector (Contd...)

we obtain

```
last=0
for i=1:q
    first=last+1
    last=first+m.vec(i)-1
    y(first:last)=A(first:last,:)x+y(first:last)
end
```

Each time through the loop an "ordinary" gaxpy is performed so Algorithms 1.1.3 and 1.1.4 apply.

Another way to block the gaxpy computation is to partition A and x as follows:

$$A = \begin{array}{c} [A_1 \quad , \ldots , \quad A_r] \\ \quad n_1 \qquad\qquad\quad n_r \end{array} \qquad x = \begin{array}{c} \begin{bmatrix} x_1 \\ \vdots \\ x_r \end{bmatrix} \end{array} \begin{array}{c} n_1 \\ \\ n_r \end{array} .$$

Block Matrix Times Vector (Contd...)

In this case we refer to A_j as the jth block column of A. If $n.vec = (n_1, \dots, n_r)$ is the vector of block column widths, then from

$$y = [A_1, \dots, A_r] \begin{bmatrix} x_1 \\ \vdots \\ x_r \end{bmatrix} + y = \sum_{j=1}^{r} A_j x_j + y$$

we obtain

```
last=0
for j=1:r
    first=last+1
    last=first+n.vec(j)-1
    y=A(:,first:last)x(first:last)+y
end
```

Again, the gaxpy's performed each time through the loop can be carried out with Algorithm 1.1.3 or 1.1.4.

Block Matrix Multiplication

Just as ordinary, scalar-level matrix multiplication can be arranged in several possible ways, so can the multiplication of block matrices. Different blockings for A, B, and C can set the stage for block versions of the dot product, saxpy, and outer product algorithms of §1.1. To illustrate this with a minimum of subscript clutter, we assume that these three matrices are all n-by-n and that $n = N\ell$ where N and ℓ are positive integers.

If $A = (A_{\alpha\beta})$, $B = (B_{\alpha\beta})$, and $C = (C_{\alpha\beta})$ are N-by-N block matrices with ℓ-by-ℓ blocks, then from Theorem 1.3.3

$$C_{\alpha\beta} = \sum_{\gamma=1}^{N} A_{\alpha\gamma} B_{\gamma\beta} + C_{\alpha\beta} \quad \alpha = 1 : N, \quad \beta = 1 : N.$$

Block Matrix Multiplication (Contd...)

If we organize a matrix multiplication procedure around this summation, then we obtain a block analog of Algorithm 1.1.5:

```
for $\alpha = 1 : N$
    $i = (a - 1)\ell + 1 : a\ell$
    for $\beta = 1 : N$
        $j = (\beta - 1)\ell + 1 : \beta\ell$
        for $\gamma = 1 : N$
            $k = (\gamma - 1)\ell + 1 : \gamma\ell$
            $C(i, j) = A(i, k)B(k, j) + C(i, j)$
        end
    end
end
```

Note that if $\ell = 1$, then $\alpha \equiv i$, $\beta \equiv j$, and $\gamma \equiv k$ and we revert to Algorithm 1.1.5.

Block Matrix Multiplication (Contd...)

To obtain a block saxpy matrix multiply, we write $C = AB + C$ as

$$[C_1, \dots, C_N] = [A_1, \dots, A_N] \begin{bmatrix} B_{11} & \cdots & B_{1N} \\ \vdots & \ddots & \vdots \\ B_{N1} & \cdots & B_{NN} \end{bmatrix} + [C_1, \dots, C_N]$$

where $A_\alpha, C_\alpha \in \mathbb{R}^{n\times\ell}$, and $B_{\alpha\beta} \in \mathbb{R}^{\ell\times\ell}$. From this we obtain

```
for $\beta = 1 : N$
    $j = (\beta - 1)\ell + 1 : \beta\ell$
    for $\alpha = 1 : N$
        $i = (\alpha - 1)\ell + 1 : \alpha\ell$
        $C(:, j) = A(:, i)B(i, j) + C(:, j)$
    end
end
```

This is the block version of Algorithm 1.1.7.

Block Matrix Multiplication (Contd...)

A block outer product scheme results if we work with the blockings

$$A = [A_1, \dots, A_N] \qquad B = \begin{bmatrix} B_1^T \\ \vdots \\ B_N^T \end{bmatrix}$$

where $A_\gamma, B_\gamma \in \mathbb{R}^{n\times\ell}$. From Lemma 1.3.2 we have

$$C = \sum_{\gamma=1}^{N} A_\gamma B_\gamma^T + C$$

and so

```
for $\gamma = 1 : N$
    $k = (\gamma - 1)\ell + 1 : \gamma\ell$
    $C = A(:, k)B(k, :) + C$
end
```

This is the block version of Algorithm 1.1.8.

Complex Matrix Multiplication

Consider the complex matrix multiplication update

$$C_1 + iC_2 = (A_1 + iA_2)(B_1 + iB_2) + (C_1 + iC_2)$$

where all the matrices are real and $i^2 = -1$. Comparing the real and imaginary parts we find

$$\begin{aligned} C_1 &= A_1 B_1 - A_2 B_2 + C_1 \\ C_2 &= A_1 B_2 + A_2 B_1 + C_2 \end{aligned}$$

and this can be expressed as follows:

$$\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} A_1 & -A_2 \\ A_2 & A_1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} + \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}.$$

This suggests how real matrix software might be applied to solve complex matrix problems. The only snag is that the explicit formation of

$$\overline{A} = \begin{bmatrix} A_1 & -A_2 \\ A_2 & A_1 \end{bmatrix}.$$

requires the "double storage" of the matrices A_1 and A_2.

A Divide and Conquer Matrix Multiplication

We conclude this section with a completely different approach to the matrix-matrix multiplication problem. The starting point in the discussion is the 2-by-2 block matrix multiplication

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

where each block is square.

In the ordinary algorithm, $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$. There are 8 multiplies and 4 adds.

A Divide and Conquer Matrix Multiplication (Contd...)

Strassen (1969) has shown how to compute C with just 7 multiplies and 18 adds:

$$\begin{aligned}
P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
P_2 &= (A_{21} + A_{22})B_{11} \\
P_3 &= A_{11}(B_{12} - B_{22}) \\
P_4 &= A_{22}(B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{12})B_{22} \\
P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
C_{11} &= P_1 + P_4 - P_5 + P_7 \\
C_{12} &= P_3 + P_5 \\
C_{21} &= P_2 + P_4 \\
C_{22} &= P_1 + P_3 - P_2 + P_6.
\end{aligned}$$

A Divide and Conquer Matrix Multiplication (Contd...)

These equations are easily confirmed by substitution. Suppose $n = 2m$ so that the blocks are m-by-m. Counting adds and multiplies in the computation $C = AB$ we find that conventional matrix multiplication involves $(2m)^3$ multiplies and $(2m)^3 - (2m)^2$ adds.

Counting adds and multiplies in the computation $C = AB$ we find that conventional matrix multiplication involves $(2m)^3$ multiplies and $(2m)^3 - (2m)^2$ adds. In contrast, if Strassen's algorithm is applied with conventional multiplication at the block level, then $7m^3$ multiplies and $7m^3 + 11m^2$ adds are required.

If $m \gg 1$, then the Strassen method involves about 7/8ths the arithmetic of the fully conventional algorithm.

A Divide and Conquer Matrix Multiplication (Contd...)

Now recognize that we can recur on the Strassen idea. In particular, we can apply the Strassen algorithm to each of the half-sized block multiplications associated with the P_i.

Thus, if the original A and B are n-by-n and $n = 2^q$, then we can repeatedly apply the Strassen multiplication algorithm.

At the bottom "level," the blocks are 1-by-1. Of course, there is no need to recur down to the $n = 1$ level.

When the block size gets sufficiently small, $(n \leq n_{\text{min}})$, it may be sensible to use conventional matrix multiplication when finding the P_i.

A Divide and Conquer Matrix Multiplication (Contd...)

Here is the overall procedure:

Algorithm 1.3.1 (Strassen Multiplication) Suppose $n = 2^q$ and that $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times n}$. If $n_{\min} = 2^d$ with $d \leq q$, then this algorithm computes $C = AB$ by applying Strassen procedure recursively $q - d$ times.

```
function: $C = strass(A, B, n, n_{\min})$
if $n \leq n_{\min}$ then
    $C = AB$
else
    $m = n/2; u = 1 : m; v = m + 1 : n;$
    $P_1 = strass(A(u,u) + A(v,v), B(u,u) + B(v,v), m, n_{\min})$
    $P_2 = strass(A(v,u) + A(v,v), B(u,u), m, n_{\min})$
    $P_3 = strass(A(u,u), B(u,v) - B(v,v), m, n_{\min})$
    $P_4 = strass(A(v,v), B(v,u) - B(u,u), m, n_{\min})$
    $P_5 = strass(A(u,u) + A(u,v), B(v,v), m, n_{\min})$
    $P_6 = strass(A(v,u) - A(u,u), B(u,u) + B(u,v), m, n_{\min})$
    $P_7 = strass(A(u,v) - A(v,v), B(v,u) + B(v,v), m, n_{\min})$
    $C(u,u) = P_1 + P_4 - P_5 + P_7$
    $C(u,v) = P_3 + P_5$
    $C(v,u) = P_2 + P_4$
    $C(v,v) = P_1 + P_3 - P_2 + P_6$
end
```

A Divide and Conquer Matrix Multiplication (Contd...)

Unlike any of our previous algorithms strass is recursive, meaning that it calls itself. Divide and conquer algorithms are often best described in this manner.

We have presented this algorithm in the style of a MATLAB function so that the recursive calls can be stated with precision.

The amount of arithmetic associated with strass is a complicated function of n and n_{min}.

If $n_{\text{min}} \gg 1$, then it suffices to count multiplications as the number of additions is roughly the same.

A Divide and Conquer Matrix Multiplication (Contd...)

If we just count the multiplications, then it suffices to examine the deepest level of the recursion as that is where all the multiplications occur.

In strass there are $q - d$ subdivisions and thus, 7^{q-d} conventional matrix-matrix multiplications to perform.

These multiplications have size n_{min} and thus strass involves about $s = (2^d)^3 7^{q-d}$ multiplications compared to $c = (2^q)^3$, the number of multiplications in the conventional approach.

Notice that

$$\frac{s}{c} = \left(\frac{2^d}{2^q}\right)^3 7^{q-d} = \left(\frac{7}{8}\right)^{q-d}.$$

A Divide and Conquer Matrix Multiplication (Contd...)

If $d = 0$, i.e., we recur on down to the 1-by-1 level, then

$$s = \left(\frac{7}{8}\right)^q c = 7^q = n^{\log_2 7} \approx n^{2.807}.$$

Thus, asymptotically, the number of multiplications in the Strassen procedure is $O(n^{2.807})$.

However, the number of additions (relative to the number of multiplications) becomes significant as n_{min} gets small.

Example 4.

If $n = 1024$ and $n_{\text{min}} = 64$, then strass involves $(7/8)^{10-6} \approx .6$ the arithmetic of the conventional algorithm.

Problems

1. Generalize (1.3.3) so that it can handle the variable block-size problem covered by Theorem 1.3.3.
2. Generalize (1.3.4) and (1.3.5) so that they can handle the variable block-size case.
3. Adapt strass so that it can handle square matrix multiplication of any order. *Hint:* If the "current" A has odd dimension, append a zero row and column.
4. Prove that if
$$A = \begin{bmatrix} A_{11} & \cdots & A_{1r} \\ \vdots & \ddots & \vdots \\ A_{q1} & \cdots & A_{qr} \end{bmatrix}$$
is a blocking of the matrix A, then
$$A^T = \begin{bmatrix} A_{11}^T & \cdots & A_{q1}^T \\ \vdots & \ddots & \vdots \\ A_{1r}^T & \cdots & A_{qr}^T \end{bmatrix}.$$

Problems (Contd...)

5. Suppose n is even and define the following function from $\mathbb{R}^n$ to $\mathbb{R}$:

$$f(x) = x(1:2:n)^T x(2:n) = \sum_{i=1}^{n/2} x_{2i-1} x_{2i}$$

   (a) Show that if $x, y \in \mathbb{R}^n$ then

$$x^T y = \sum_{i=1}^{n/2} (x_{2i-1} + y_{2i})(x_{2i} + y_{2i-1}) - f(x) - f(y)$$

   (b) Now consider the n-by-n matrix multiplication $C = AB$. Give an algorithm for computing this product that requires $n^3/2$ multiplies once f is applied to the rows of A and the columns of B.

Problems (Contd...)

6. Prove Lemma 1.3.2 for general s. *Hint.* Set

$$p_T = p_1 + \cdots + p_{\gamma-1} \qquad \gamma = 1 : s+1$$

and show that

$$c_{ij} = \sum_{\gamma=1}^{s} \sum_{k=p_\gamma+1}^{p_{\mu+1}} a_{ik} b_{kj}.$$

7. Use Lemmas 1.3.1 and 1.3.2 to prove Theorem 1.3.3. In particular, set

$$A_\gamma = \begin{bmatrix} A_{1\gamma} \\ \vdots \\ A_{q\gamma} \end{bmatrix} \quad \text{and} \quad B_\gamma = \begin{bmatrix} B_{\gamma 1} & \cdots & B_{\gamma r} \end{bmatrix}$$

and note from Lemma 1.3.2 that

$$C = \sum_{\gamma=1}^{s} A_\gamma B_\gamma.$$

Now analyze each $A_\gamma B_\gamma$ with the help of Lemma 1.3.1.

Reference Books

1. Gene H. Golub and Charles F. Van Loan, Matrix Computations, 3rd Edition, Hindustan book agency, 2007.
2. A.R. Gourlay and G.A. Watson, Computational methods for matrix eigen problems, John Wiley & Sons, New York, 1973.
3. W.W. Hager, Applied numerical algebra, Prentice-Hall, Englewood Cliffs, N.J, 1988.
4. D.S. Watkins, Fundamentals of matrix computations, John Wiley and sons, N.Y, 1991.
5. C.F. Van Loan, Introduction to scientific computing: A Matrix vector approach using Matlab, Prentice-Hall, Upper Saddle River, N.J, 1997.