Basic Algorithms and Notation

P. Sam Johnson

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

Introduction

The proper study of matrix computations begins with the study of the matrix-matrix multiplication problem.

Although this problem is simple mathematically it is very rich from the computational point of view. We begin by looking at the several ways that the matrix multiplication problem can be organized.

The "language" of partitioned matrices is established and used to characterize several linear algebraic "levels" of computation.

Basic Algorithms and Notation

Matrix computations are built upon a hierarchy of linear algebraic operations.

Dot products involve the scalar operations of addition and multiplication. Matrix-vector multiplication is made up of dot products.

Matrix-matrix multiplication amounts to a collection of matrix-vector products. All of these operations can be described in algorithmic form or in the language of linear algebra. Our primary objective in this section is to show how these two styles of expression complement each another.

Along the way we pick up notation and acquaint the reader with the kind of thinking that underpins the matrix computation area.

The discussion revolves around the matrix multiplication problem, a computation that can be organized in several ways.

Matrix Notation

Let $\mathbb{R}$ denote the set of real numbers. We denote the vector space of all m-by-n real matrices by $\mathbb{R}^{m\times n}$:

$$A \in \mathbb{R}^{m\times n} \iff A = (a_{ij}) = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \quad a_{ij} \in \mathbb{R}.$$

If a capital letter is used to denote a matrix (e.g. A, B, Δ), then the corresponding lower case letter with subscript ij refers to the (i, j) entry (e.g., $a_{ij}, b_{ij}, \delta_{ij}$). As appropriate, we also use the notation $[A]_{ij}$ and $A(i, j)$ to designate the matrix elements.

Matrix Operations

Basic matrix operations include transposition ($\mathbb{R}^{m\times n} \to \mathbb{R}^{n\times m}$),

$$C = A^T \implies c_{ij} = a_{ji},$$

addition ($\mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$),

$$C = A + B \implies c_{ij} = a_{ij} + b_{ij},$$

scalar-matrix multiplication, ($\mathbb{R} \times \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$),

$$C = \alpha A \implies c_{ij} = \alpha a_{ij},$$

and matrix-matrix multiplication ($\mathbb{R}^{m\times p} \times \mathbb{R}^{p\times n} \to \mathbb{R}^{m\times n}$),

$$C = AB \implies c_{ij} = \sum_{k=1}^{r} a_{ik} b_{kj}.$$

These are the building blocks of matrix computations.

Vector Notation

Let $\mathbb{R}^n$ denote the vector space of real n-vectors:

$$x \in \mathbb{R}^n \iff x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad x_i \in \mathbb{R}.$$

We refer to x_i as the ith component of x. Depending upon context, the alternative notations $[x]_i$ and $x(i)$ are sometimes used.

Notice that we are identifying $\mathbb{R}^n$ with $\mathbb{R}^{n\times 1}$ and so the members of $\mathbb{R}^n$ are *column* vectors. On the other hand, the elements of $\mathbb{R}^{1\times n}$ vectors:

$$x \in \mathbb{R}^{1\times n} \iff x = (x_1, \ldots, x_n).$$

If x is a column vector, then $y = x^T$ is a row vector.

Vector Operations

Assume $a \in \mathbb{R}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^n$. Basic vector operations include scalar-vector multiplication,

$$z = ax \Longrightarrow z_i = ax_i,$$

vector addition,

$$z = x + y \Longrightarrow z_i = x_i + y_i,$$

the dot product (or inner product),

$$c = x^T y \Longrightarrow c = \sum_{i=1}^{n} x_i y_i,$$

and vector multiply (or the Hadamard product)

$$z = x * y \Longrightarrow z_i = x_i y_i.$$

Vector Operations

Another very important operation which we write in "update form" is the saxpy:

$$y = ax + y \Longrightarrow y_i = ax_i + y_i$$

Here, the symbol "=" is being used to denote assignment, not mathematical equality. The vector y is being updated.

The name "saxpy" is used in LAPACK, a software package that implements many of the algorithms in this book.

One can think of "saxpy" as a mnemonic for "scalar a x plus y."

The Computation of Dot Products and Saxpys

We have chosen to express algorithms in a stylized version of the MATLAB language. MATLAB is a powerful interactive system that is ideal for matrix computation work. We gradually introduce our stylized MATLAB notation in this chapter beginning with an algorithm for computing dot products.

Algorithm 1.1.1 (Dot Product) If $x, y \in \mathbb{R}^n$, then this algorithm computes their dot product $c = x^T y$.

$c = 0$
for i=1:n
    $c = c + x(i)y(i)$
end

The dot product of two n-vectors involves n multiplications and n additions. It is an "O(n)" operation, meaning that the amount of work is linear in the dimension. The saxpy computation is also an $O(n)$ operation, but it returns a vector instead of a scalar.

The Computation of Dot Products and Saxpys (Contd...)

Algorithm 1.1.2 (Saxpy) If $x, y \in \mathbb{R}^n$ and $a \in \mathbb{R}$, then this algorithm overwrites y with $ax + y$.

for i=1:n

$y(i) = ax(i) + y(i)$

end

It must be stressed that the algorithms are encapsulations of critical computational ideas and not "production codes."

Matrix-Vector Multiplication and the Gaxpy

Suppose $A \in \mathbb{R}^{m \times n}$ and that we wish to compute the update

$$y = Ax + y$$

where $x \in \mathbb{R}^n$ and $y = \mathbb{R}^m$ are given. This generalized saxpy operation is referred to as a gaxpy. A standard way that this computation proceeds is to update the components one at a time:

$$y_i = \sum_{j=1}^{n} a_{ij} x_j + y_i \qquad i = 1 : m.$$

Matrix-Vector Multiplication and the Gaxpy (Contd...)

This gives the following algorithm.

Algorithm 1.1.3 (Gaxpy: Row Version) If $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, then this algorithm overwrites y with $Ax + y$.

for i=1:m
  for i=1:n
    $y(i) = A(i,j)x(j) + y(i)$
  end
end

Matrix-Vector Multiplication and the Gaxpy (Contd...)

An alternative algorithm results if we regard Ax as a linear combination of A's columns, e.g.,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 7 + 2 \cdot 8 \\ 3 \cdot 7 + 4 \cdot 8 \\ 5 \cdot 7 + 6 \cdot 8 \end{bmatrix} = 7 \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} + 8 \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 23 \\ 53 \\ 83 \end{bmatrix}.$$

Algorithm 1.1.4 (Gaxpy: Column Version) If $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, then this algorithm overwrites y with $Ax + y$.

for j=1:n
  for i=1:m
    $y(i) = A(i,j)x(j) + y(i)$
  end
end

Matrix-Vector Multiplication and the Gaxpy (Contd...)

Note that the inner loop in either gaxpy algorithm carries out a saxpy operation.

The column version was derived by rethinking what matrix-vector multiplication "means" at the vector level, but it could also have been obtained simply by interchanging the order of the loops in the row version.

In matrix computations, it is important to relate loop interchanges to the underlying linear algebra.

Algorithms 1.1.3 and 1.1.4 access the data in A by row and by column respectively.

To highlight these orientations more clearly we introduce the language of partitioned matrices.

Partitioning a Matrix into Rows and Columns

From the row point of view, a matrix is a stack of row vectors:

$$A \in \mathbb{R}^{m \times n} \iff A = \begin{bmatrix} r_1^T \\ \vdots \\ r_m^T \end{bmatrix} \quad r_k \in \mathbb{R}^n. \tag{1}$$

This is called a row partition of A. Thus, if we row partition

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix},$$

then we are choosing to think of A as a collection of rows with

$$r_1^T = \begin{bmatrix} 1 & 2 \end{bmatrix}, r_2^T = \begin{bmatrix} 3 & 4 \end{bmatrix}, r_3^T = \begin{bmatrix} 5 & 6 \end{bmatrix}.$$

Matrix-Vector Multiplication and the Gaxpy (Contd...)

With the row partitioning (1) Algorithm 1.1.3 can be expressed as follows:

for i=1:m

$y_i = r_i^T x + y(i)$

end

Partitioning a Matrix into Rows and Columns (Contd...)

Alternatively, a matrix is a collection of column vectors:

$$A \in \mathbb{R}^{m \times n} \iff A = [c_1, \ldots, c_n], \quad c_k \in \mathbb{R}^m. \tag{2}$$

We refer to this as a column partition of A.

In the 3-by-2 example above, we thus would set c_1 and c_2 to be the first and second columns of A respectively:

$$c_1 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \qquad c_2 = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}.$$

Partitioning a Matrix into Rows and Columns (Contd...)

With (2) we see that Algorithm 1.1.4 is a saxpy procedure that accesses A by columns:

for j=1:n

$y = x_j c_j + y$

end

In this context appreciate y as a running vector sum that undergoes repeated saxpy updates.

The Colon Notation

A handy way to specify a column or row of a matrix is with the "colon" notation. If $A \in \mathbb{R}^{m \times n}$, then $A(k,:)$ designates the kth row, i.e.,

$$A(k,:) = [a_{k1}, \ldots, a_{kn}].$$

The kth column is specified by

$$A(:,k) = \begin{bmatrix} a_{1k} \\ \vdots \\ a_{mk} \end{bmatrix}.$$

The Colon Notation

With these conventions we can rewrite Algorithms 1.1.3 and 1.1.4 as

for i=1:m
$y(i) = A(i,:)x + y(i)$
end

and

for j=1:n
$y = x(j)A(:,j) + y$
end

respectively. With the colon notation we are able to suppress iteration details. This frees us to think at the vector level and focus on larger computational issues.

The Outer Product Update

As a preliminary application of the colon notation, we use it to understand the outer product update

$$A = A + xy^T, \quad A \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^m, y \in \mathbb{R}^n.$$

The outer product operation xy^T "looks funny" but is perfectly legal, e.g.,

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 4 & 5 \end{bmatrix} = \begin{bmatrix} 4 & 5 \\ 8 & 10 \\ 12 & 15 \end{bmatrix}.$$

The Outer Product Update

This is because xy^T is the product of two "skinny" matrices and the number of columns in the left matrix x equals the number of rows in the right matrix y^T. The entries in the outer product update are prescribed by

for i=1:m
    for j=1:n
        $a_{ij} = a_{ij} + x_i y_j$
    end
end

The Outer Product Update (Contd...)

The mission of the j loop is to add a multiple of y^T to the i-th row of A, i.e.,

for i=1:m

$$A(i,:) = A(i,:) + x(i)y^T$$

end

On the other hand, if we make the i-loop the inner loop, then its task is to add a multiple of x to the jth column of A :

for j=1:n

$$A(:,j) = A(:,j) + y(j)x$$

end

Note that both outer product algorithms amount to a set of saxpy updates.

Matrix-Matrix Multiplication

Consider the 2-by-2 matrix-matrix multiplication AB. In the dot product formulation each entry is computed as a dot product:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix}.$$

In the saxpy version each column in the product is regarded as a linear combination of columns of A :

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \left[5 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 7 \begin{bmatrix} 2 \\ 4 \end{bmatrix}, 6 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 8 \begin{bmatrix} 2 \\ 4 \end{bmatrix} \right].$$

Matrix-Matrix Multiplication

Finally, in the outer product version, the result is regarded as the sum of outer products:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \begin{bmatrix} 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix} \begin{bmatrix} 7 & 8 \end{bmatrix}.$$

Although equivalent mathematically, it turns out that these versions of matrix multiplication can have very different levels of performance because of their memory traffic properties. This matter is pursued in §1.4. For now, it is worth detailing the above three approaches to matrix multiplication because it gives us a chance to review notation and to practice thinking at different linear algebraic levels.

Scalar-Level Specifications

To fix the discussion we focus on the following matrix multiplication update:

$$C = AB + C \qquad A \in \mathbb{R}^{m \times p}, B \in \mathbb{R}^{p \times n}, C \in \mathbb{R}^{m \times n}.$$

The starting point is the familiar triply-nested loop algorithm:

Scalar-Level Specifications

Algorithm 1.1.5 (Matrix Multiplication: *ijk* Variant) If $A \in \mathbb{R}^{m \times p}$, $B \in \mathbb{R}^{p \times n}$, and $C \in \mathbb{R}^{m \times n}$ are given, then this algorithm overwrites C with $AB + C$.

```
for i=1:m
    for j=1:n
        for k=1:p
            C(i,j) = A(i,k)B(k,j) + C(i,j)
        end
    end
end
```

This is the "*ijk* variant" because we identify the rows of C (and A) with i, the columns of C (and B) with j, and the summation index with k.

Scalar-Level Specifications (Contd...)

We consider the update $C = AB + C$ instead of just $C = AB$ for two reasons. We do not have to bother with $C = 0$ initializations and updates of the form $C = AB + C$ arise more frequently in practice.

The three loops in the matrix multiplication update can be arbitrarily ordered giving $3! = 6$ variations. Thus,

```
for j=1:n
    for k=1:p
        for i=1:m
            C(i,j) = A(i,k)B(k,j) + C(i,j)
        end
    end
end
```

is the *jki* variant.

Scalar-Level Specifications (Contd...)

Each of the six possibilities ($ijk, jik, ikj, jki, kij, kji$) features an inner loop operation (dot product or saxpy) and has its own pattern of data flow.

For example, in the *ijk* variant, the inner loop oversees a dot product that requires access to a row of A and a column of B. The *jki* variant involves a saxpy that requires access to a column of C and a column of A. These attributes are summarized in Table 1 along with an interpretation of what is going on when the middle and inner loop are considered together.

Each variant involves the same amount of floating point arithmetic, but accesses the A, B, and C data differently.

Scalar-Level Specifications (Contd...)

Loop Order	Inner Loop	Middle Loop	Inner Loop Data Access
ijk	dot	vector $\times$ matrix	A by row, B by column
jik	dot	matrix $\times$ vector	A by row, B by column
ikj	saxpy	row gaxpy	B by row, C by row
jki	saxpy	column gaxpy	A by column, C by column
kij	saxpy	row outer product	B by row, C by row
kji	saxpy	column outer product	A by column, C by column

Table: Matrix Multiplication: Loop Orderings and Properties

A Dot Product Formulation

The usual matrix multiplication procedure regards AB as an array of dot products to be computed one at a time in left-to-right, top-to-bottom order. This is the idea behind Algorithm 1.1.5. Using the colon notation we can highlight this dot-product formulation:

Algorithm 1.1.6 (Matrix Multiplication: Dot Product Version) If $A \in \mathbb{R}^{m\times p}$, $B \in \mathbb{R}^{p\times n}$, and $C \in \mathbb{R}^{m\times n}$ are given, then this algorithm overwrites C with $AB + C$.

for i=1:m
  for j=1:n
    $C(i,j) = A(i,:)B(:,j) + C(i,j)$
  end
end

A Dot Product Formulation (Contd...)

In the language of partitioned matrices, if

$$A = \begin{bmatrix} a_1^T \\ \vdots \\ a_m^T \end{bmatrix} \qquad a_k \in \mathbb{R}^p$$

and

$$B = [b_1, \ldots, b_n] \quad b_k \in \mathbb{R}^p$$

then Algorithm 1.1.6 has this interpretation:

for i=1:m
  for j=1:n
    $c_{ij} = a_i^T b_j + c_{ij}$
  end
end

A Dot Product Formulation (Contd...)

Note that the "mission" of the j-loop is to compute the ith row of the update. To emphasize this we could write

for i=1:m

$$c_i^T = a_i^T B + c_i^T$$

end

where

$$C = \begin{bmatrix} c_1^T \\ \vdots \\ c_m^T \end{bmatrix}$$

is a row partitioning of C.

A Dot Product Formulation (Contd...)

To say the same thing with the colon notation we write

for i=1:m

$C(i,:) = A(i,:)B + C(i,:)$

end

Either way we see that the inner two loops of the *ijk* variant define a row-oriented gaxpy operation.

A Saxpy Formulation

Suppose A and C are column-partitioned as follows

$$A = [a_1, \ldots, a_p] \quad a_j \in \mathbb{R}^m$$

$$C = [c_1, \ldots, c_n] \quad c_j \in \mathbb{R}^m.$$

By comparing jth columns in $C = AB + C$ we see that

$$c_j = \sum_{k=1}^{p} b_{kj} a_k + c_j, \quad j = 1 : n.$$

A Saxpy Formulation

These vector sums can be put together with a sequence of saxpy updates.

Algorithm 1.1.7 (Matrix Multiplication: Saxpy Version) If the matrices $A \in \mathbb{R}^{m\times p}$, $B \in \mathbb{R}^{p\times n}$, and $C \in \mathbb{R}^{m\times n}$ are given, then this algorithm overwrites C with $AB + C$.

for j=1:n
  for k=1:p
    $C(:,j) = A(:,k)B(k,j) + C(:,j)$
  end
end

Note that the k-loop oversees a gaxpy operation:

for j=1:n
  $C(:,j) = AB(:,j) + C(:,j)$
end

An Outer Product Formulation

Consider the *kij* variant of Algorithm 1.1.5:

for k=1:p
  for j=1:n
    for i=1:m
      $C(i,j) = A(i,k)B(k,j) + C(i,j)$
    end
  end
end

An Outer Product Formulation

The inner two loops oversee the outer product update $C = a_k b_k^T + C$ where

$$A = [a_1, \ldots, a_p] \quad \text{and} \quad B = \begin{bmatrix} b_1^T \\ \vdots \\ b_p^T \end{bmatrix} \tag{3}$$

with $a_k \in \mathbb{R}^m$ and $b_k \in \mathbb{R}^n$. We therefore obtain

Algorithm 1.1.8 (Matrix Multiplication: Outer Product Version) If $A \in \mathbb{R}^{m \times p}$, $B \in \mathbb{R}^{p \times n}$, $C \in \mathbb{R}^{m \times n}$ are given, then this algorithm overwrites C with $AB + C$.

for k=1:p
$C = A(:, k)B(k, :) + C$
end

This implementation revolves around the fact that AB is the sum of p outer products.

The Notion of "Level"

The dot product and saxpy operations are examples of "level-1" operations. Level-1 operations involve an amount of data and an amount of arithmetic that is linear in the dimension of the operation. An m-by-n outer product update or gaxpy operation involves a quadratic amount of data ($O(mn)$) and a quadratic amount of work ($O(mn)$). They are examples of " "level-2" operations.

The matrix update $C = AB + C$ is a "level-3" operation. Level-3 operations involve a quadratic amount of data and a cubic amount of work. If A, B, and C are n-by-n matrices, then $C = AB + C$ involves $O(n^2)$ matrix entries and $O(n^3)$ arithmetic operations.

The Notion of "Level"

The design of matrix algorithms that are rich in high-level linear algebra operations is a recurring theme in the book.

For example, a high performance linear equation solver may require a level-3 organization of Gaussian elimination.

This requires some algorithmic rethinking because that method is usually specified in level-1 terms, e.g., "multiply row 1 by a scalar and add the result to row 2."

A Note on Matrix Equations

In striving to understand matrix multiplication via outer products, we essentially established the matrix equation

$$AB = \sum_{k=1}^{p} a_k b_k^T$$

where the a_k and b_k are defined by the partitionings in (3).

Numerous matrix equations are developed in subsequent chapters. Sometimes they are established algorithmically like the above outer product expansion and other times they are proved at the ij-component level. As an example of the latter, we prove an important result that characterizes transposes of products.

A Note on Matrix Equations (Contd...)

Theorem 1.

If $A \in \mathbb{R}^{m\times p}$ and $B \in \mathbb{R}^{p\times n}$, then $(AB)^T = B^T A^T$.

Proof: If $C = (AB)^T$, then

$$c_{ij} = [(AB)^T]_{ij} = [AB]_{ji} = \sum_{k=1}^{p} a_{jk} b_{ki}.$$

On the other hand, if $D = B^T A^T$, then

$$d_{ij} = [B^T A^T]_{ij} = \sum_{k=1}^{p} [B^T]_{ik} [A^T]_{kj} = \sum_{k=1}^{p} b_{ki} a_{jk}.$$

Since $c_{ij} = d_{ij}$ for all i and j, it follows that $C = D$.

Scalar-level proofs such as this one are usually not very insightful. However, they are sometimes the only way to proceed.

Complex Matrices

From time to time computations that involve complex matrices are discussed. The vector space of m-by-n complex matrices is designated by $\mathbb{C}^{m\times n}$. The scaling, addition, and multiplication of complex matrices corresponds exactly to the real case. However, transposition becomes conjugate transposition:

$$C = A^H \Longrightarrow c_{ij} = \overline{a}_{ji}.$$

The vector space of complex n-vectors is designated by $\mathbb{C}^n$. The dot product of complex n-vectors x and y is prescribed by

$$s = x^H y = \sum_{i=1}^{n} \overline{x}_i y_i.$$

Finally, if $A = B + iC \in \mathbb{C}^{m\times n}$, then we designate the real and imaginary parts of A by $Re(A) = B$ and $Im(A) = C$ respectively.

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.