Matrix Multiplication Problems
Part - 1

P. Sam Johnson

September 15, 2014

Overview

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**.

- The **several ways** that the matrix multiplication problem is organized.
- 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 is to show how these two styles of expression complement each another.

Matrix Notation

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

$$A \in \mathbb{R}^{m\times n} \Leftrightarrow A = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \qquad a_{ij} \in \mathbb{R}.$$

Capital letters (e.g. A, B) are used to denote matrices whereas the corresponding lower case letters (e.g. a_{ij}, b_{ij}) refer to entries of the matrices.

Greek letters (e.g. α, β) are usually denoted for (real) scalars.

Matrix Operations

Basic matrix operations include

1. **addition** : $(A, B) \mapsto A + B$.
2. **scalar-matrix multiplication** : $(\alpha, A) \mapsto \alpha A$.
3. **matrix-matrix multiplication** : $(A, B) \mapsto AB$.
4. **transposition** : $A \mapsto A^T$.

These are the **building blocks of matrix computations**.

Vector Space : The set of all the n-tuples with real entries

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

$$x \in \mathbb{R}^n \Leftrightarrow x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad x_i \in \mathbb{R}.$$

We refer to x_i as the ith component of x.

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 other hand, the elements of $\mathbb{R}^{1\times n}$ are row vectors.

$$x \in \mathbb{R}^{1\times n} \Leftrightarrow x = (x_1, x_2, \ldots, x_n).$$

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

Vector Operations

Assume $\alpha \in \mathbb{R}, x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$. Basic vector operations include

1. **scalar-vector multiplication** : $(\alpha, x) \mapsto \alpha x$

$$z = \alpha x \quad \Longrightarrow \quad z_i = \alpha x_i,$$

2. **vector addition** : $(x, y) \mapsto x + y$

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

3. the **dot product** (or **inner product**) : $(x, y) \mapsto x^T y$

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

4. **vector multiply** (or the **Hadamard product**) : $(x, y) \mapsto x. * y$

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

Another Important Vector Operation

Another very important operation which we write in "**update form**" is the **saxpy**. It means "**S**calar a x plus y.

$$y = ax + y \quad \Rightarrow \quad y_i = ax_i + y_i.$$

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 the course.

Algorithm : Dot Product

We have chosen to express algorithms in a stylized version of the MATLAB language.

Algorithm (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.

Algorithm : Saxpy Computation

The saxpy computation is also an $O(n)$ operation, but it returns a vector instead of a scalar.

Algorithm (Saxpy)

If $x, y \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$, then this algorithm overwrites y with $\alpha x + y$.

for $i = 1 : n$

$\quad y(i) = \alpha x(i) + y(i)$

end

Generalized saxpy, called 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 $y \in \mathbb{R}^m$ are given.

This **g**eneralized **saxpy** operation is referred to as a **gaxpy**.

A standard way that this computation proceeds is to update the components on at a time:

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

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

Algorithm : Gaxpy - Column version

Algorithm (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

Partitioning a Matrix into Rows and Columns

Algorithms [Gaxpy: Row version] and [Gaxpy: Column version] access the data in A by row and by column respectively.

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

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

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

This is called a **row partition** of A.

Example : Row Partition

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

With the row partitioning, Algorithm [Gaxpy: Row version] can be expressed as follows:

for $i = 1 : m$
$\quad y_i = r_i^T x + y(i)$
end

Column Partition

Alternatively, a matrix is a collection of columns vectors:

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

We refer to this as a **column partition** of A.

With (2) we see that Algorithm [Gaxpy : Column Version] is a saxpy procedure that accesses A by column.

for $j = 1 : n$
    $y = x_j c_j + y$
end

"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 & $A(:,k)$ designates the kth column. With this conventions we can rewrite Algorithms [Gaxpy : Row and Column Versions] as

$$\begin{aligned}
&\text{for } i = 1 : m \\
&\qquad y(i) = A(i,:)x + y(i) \\
&\text{end}
\end{aligned}$$

and

$$\begin{aligned}
&\text{for } j = 1 : n \\
&\qquad y = x(j)A(:,j) + y \\
&\text{end}
\end{aligned}$$

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.

Outer Product Update

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

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

The outer product operation xy^T "looks funny" but is perfectly legal. For example

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

is the product of two "skinny" matrices. 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

Matrix-Matrix Multiplication

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

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

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.

Scalar-Level Specifications

We focus on the following matrix multiplication update

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

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

Algorithm (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.

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 $3! = 6$ vairations.

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

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

Matrix Multiplication : Dot Product Version

Using the colon notation we can highlight this dot-product formulation.

Algorithm (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

Matrix Multiplication : Saxpy Version

Algorithm (Matrix Multiplicationm : 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

Matrix Multiplication : Outer Product Version

Algorithm (Matrix Multiplication : Outer 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 $k = 1 : p$
    $C = A(:, k)B(k, :) + C$
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 and amount of data and an amount of arithmetic that is linear in the dimension of the operation. An $m \times n$ 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 $n \times n$ matrices, then $C = AB + C$ involves $O(n^2)$ matrix entries and $O(n^3)$ arithmetic operations.

Numerous matrix equations are established algorithmically like athe 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.

Theorem

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 $C = D$. $\square$

Computations that involve complex matrices

The vector space of $m \times n$ 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 :**

$$C = A^H \quad \Rightarrow \quad c_{ij} = \bar{a}_{ji}.$$

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

References

- **Gene H. Golub and Charles F. Van Loan**, *Matrix Computations*, Third Edition, Hindustan Book Agency, 2007.
- **D.S. Watkins**, *Fundamentals of Matrix Computations*, John Wiley & Sons, New York, 1991.