Chapter 1

Linear Algebra

1.1 Example

Consider the following problem: fit a polynomial of degree $n - 1$ to the following points:

$$(\chi_1, \eta_1), (\chi_2, \eta_2), \ldots, (\chi_n, \eta_n)$$

where $\chi_i$ and $\eta_i$ are real numbers $(\chi_i, \eta_i \in \mathbf{R})$. Notice that we are looking for the coefficients of the polynomial

$$p_n(\chi) = a_0 + a_1 \chi + a_2 \chi^2 + \cdots + a_{n-1} \chi^{n-1}$$

and we know that

$$
\begin{align*}
p_n(\chi_1) &= a_0 + a_1 \chi_1 + a_2 \chi_1^2 + \cdots + a_{n-1} \chi_1^{n-1} = \eta_1 \\
p_n(\chi_2) &= a_0 + a_1 \chi_2 + a_2 \chi_2^2 + \cdots + a_{n-1} \chi_2^{n-1} = \eta_2 \\
&\vdots \\
p_n(\chi_n) &= a_0 + a_1 \chi_n + a_2 \chi_n^2 + \cdots + a_{n-1} \chi_n^{n-1} = \eta_n
\end{align*}
$$

Thus, we must solve this linear system of equations for $\{a_0, \ldots, a_{n-1}\}$.

Notice the above set of linear equations can be rewritten as

$$
\alpha_0 \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} + \alpha_1 \begin{pmatrix} \chi_1 \\ \chi_2 \\ \vdots \\ \chi_n \end{pmatrix} + \alpha_2 \begin{pmatrix} \chi_1^2 \\ \chi_2^2 \\ \vdots \\ \chi_n^2 \end{pmatrix} + \cdots + \alpha_{n-1} \begin{pmatrix} \chi_1^{n-1} \\ \chi_2^{n-1} \\ \vdots \\ \chi_n^{n-1} \end{pmatrix} = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_n \end{pmatrix}
$$

or, in matrix notation:

$$
\begin{pmatrix}
1 & \chi_1 & \chi_1^2 & \cdots & \chi_1^{n-1} \\
1 & \chi_2 & \chi_2^2 & \cdots & \chi_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \chi_n & \chi_n^2 & \cdots & \chi_n^{n-1}
\end{pmatrix}
\begin{pmatrix}
\alpha_0 \\
\alpha_1 \\
\vdots \\
\alpha_{n-1}
\end{pmatrix}
=
\begin{pmatrix}
\eta_1 \\
\eta_2 \\
\vdots \\
\eta_n
\end{pmatrix}
$$

We call the two-dimensional array

$$X = 
\begin{pmatrix}
1 & \chi_1 & \chi_1^2 & \cdots & \chi_1^{n-1} \\
1 & \chi_2 & \chi_2^2 & \cdots & \chi_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \chi_n & \chi_n^2 & \cdots & \chi_n^{n-1}
\end{pmatrix}
$$
an $n \times n$ matrix,

$$a = \begin{pmatrix}
\alpha_0 \\
\alpha_1 \\
\vdots \\
\alpha_{n-1}
\end{pmatrix}$$

the vector of unknowns, and

$$y = \begin{pmatrix}
\eta_1 \\
\eta_2 \\
\vdots \\
\eta_n
\end{pmatrix}$$

the right-hand-side vector.

We are thus left to compute $a$ from the linear equation

$$Xa = y$$

### 1.2 Vectors and Matrices

#### 1.2.1 Vectors

**Definition 1.1** A (column) vector, $x$, is the $n$-tuple of real or complex numbers

$$x = \begin{pmatrix}
\chi_1 \\
\chi_2 \\
\vdots \\
\chi_n
\end{pmatrix}$$

Here $\chi_i$ are called the components of vector $x$. We will denote the set of all vectors with real components $\mathbb{R}^n$ and with complex components $\mathbb{C}^n$.

**Note:** We will generally use lower case letters to denote column vectors.

**Definition 1.2** A row vector, $x^T$, is the $n$-tuple of real or complex numbers

$$x^T = \begin{pmatrix}
\chi_1 & \chi_2 & \cdots & \chi_n
\end{pmatrix}$$

Here $x^T$ indicates a transposed (column) vector. (We will always assume vectors are column vectors, unless transposed like this, or explicitly noted.) More about transposition later.

**Note:** Since lower case letters will be used to denote column vectors, we will use lower case letters together with the transpose $^T$ to denote row vectors.

#### 1.2.2 Matrices

**Definition 1.3** An $m \times n$ matrix, $A$, is the array

$$A = \begin{pmatrix}
\alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\
\alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\alpha_{m1} & \alpha_{m2} & \cdots & \alpha_{mn}
\end{pmatrix}$$

with $m$ rows and $n$ columns. The $(i,j)$ component, or element, of $A$ refers to $\alpha_{ij}$, which may be real or complex. The numbers $m$ and $n$ are the dimensions of $A$. If $m = n$ then the matrix is said to be square. Otherwise, it is said to be rectangular.
1.3. OPERATIONS ON VECTORS AND MATRICES

Note: We will generally use upper case letters to denote matrices.

1.3. Operations on vectors and matrices

1.3.1. Vector-vector operations

Definition 1.4 (Equality) Let \( x, y \in \mathbb{R}^n \), with \( x = \begin{pmatrix} \chi_1 \\ \chi_2 \\ \vdots \\ \chi_n \end{pmatrix} \) and \( y = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_n \end{pmatrix} \). Then \( x \) and \( y \) are said to be equal if \( \chi_i = \eta_i \), \( i = 1, \ldots, n \).

The assertion \( x = y \) will be used to denote the equality of the two vectors.

Note: We will also use the notation \( x = y \) to indicate assignment of the values of \( y \) to vector \( x \).

Definition 1.5 (Copy) Given that \( x, y \in \mathbb{R}^n \) we will use the assignment \( y = x \) to indicate that the values of the elements of \( x \) are copied to become the values of the elements of \( y \):

\[ \eta_i = \chi_i \quad i = 1, \ldots, n \]

Definition 1.6 (Scale) Let \( \alpha \in \mathbb{R} \) and \( x \in \mathbb{R}^n \). Then the scaling of the vector \( x \) by a factor \( \alpha \) is given by

\[ \alpha x = \begin{pmatrix} \chi_1 \\ \chi_2 \\ \vdots \\ \chi_n \end{pmatrix} = \begin{pmatrix} \alpha \chi_1 \\ \alpha \chi_2 \\ \vdots \\ \alpha \chi_n \end{pmatrix} \]

The scaling of a row vector is defined as the scaling of the elements of that row vector.

Definition 1.7 (AXPY) Let \( \alpha \in \mathbb{R} \) and \( x, y \in \mathbb{R}^n \). Then the scaled addition of vector \( x \) and \( y \) is given by

\[ \alpha x + y = \alpha \begin{pmatrix} \chi_1 \\ \chi_2 \\ \vdots \\ \chi_n \end{pmatrix} + \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_n \end{pmatrix} = \begin{pmatrix} \alpha \chi_1 + \eta_1 \\ \alpha \chi_2 + \eta_2 \\ \vdots \\ \alpha \chi_n + \eta_n \end{pmatrix} \]

The name AXPY comes from the operation \( y \leftarrow \alpha x + y \): \( \alpha \) times vector \( x \) plus vector \( y \).

Definition 1.8 (Dot) Given \( x, y \in \mathbb{R}^n \), the inner product (dot-product) of \( x \) and \( y \) is a real number given by

\[ \sum_{i=1}^{n} \chi_i \eta_i. \]

Once we define transposition and matrix-matrix multiplication, we will see that this also equal \( x^T y \) and \( y^T x \).

1.3.2. Matrix-vector operations

Transposition

Definition 1.9 Let \( A \in \mathbb{R}^{m \times n} \) and \( B \in \mathbb{R}^{n \times m} \). Then \( B \) is said to be the transpose of \( A \) if \( \beta_{ij} = \alpha_{ji} \) for all \( i, j = 1, \ldots, n \) and \( j, j = 1, \ldots, m \).

The transpose of a matrix will be denoted by \( A^T \).

Note: If \( B = A^T \), then \( A = B^T \).
Scaling a matrix

Let \( \alpha \in \mathbb{R}^n \) and \( A \in \mathbb{R}^{m \times n} \). Scaling a matrix is achieved by scaling the individual elements of the matrix:

\[
\beta A = \beta \begin{pmatrix}
\alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\
\alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\alpha_{m1} & \alpha_{m2} & \cdots & \alpha_{mn}
\end{pmatrix} = \begin{pmatrix}
\beta \alpha_{11} & \beta \alpha_{12} & \cdots & \beta \alpha_{1n} \\
\beta \alpha_{21} & \beta \alpha_{22} & \cdots & \beta \alpha_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\beta \alpha_{m1} & \beta \alpha_{m2} & \cdots & \beta \alpha_{mn}
\end{pmatrix}
\]

(GEMV) Matrix times a vector

In Section 1.1 we already saw an example of matrix-vector multiplication. Given matrix \( A \in \mathbb{R}^{m \times n} \), vector \( x \in \mathbb{R}^n \) and vector \( y \in \mathbb{R}^m \),

\[
y = Ax = \begin{pmatrix}
\alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\
\alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\alpha_{m1} & \alpha_{m2} & \cdots & \alpha_{mn}
\end{pmatrix} \begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}
\]

\[
y = \begin{pmatrix} \alpha_{11} x_1 + \alpha_{21} x_2 + \cdots + \alpha_{m1} x_n \\ \alpha_{12} x_1 + \alpha_{22} x_2 + \cdots + \alpha_{m2} x_n \\ \vdots \\ \alpha_{1n} x_1 + \alpha_{2n} x_2 + \cdots + \alpha_{mn} x_n \end{pmatrix}
\]

Outer product and (GER) rank-1 update

**Definition 1.10** Let \( x \in \mathbb{R}^m \) and \( y \in \mathbb{R}^n \). The outer product of \( x \) and \( y \) is given by the \( m \times n \) matrix

\[
x y^T = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \begin{pmatrix} \eta_1^T \\ \eta_2^T \\ \vdots \\ \eta_N^T \end{pmatrix} = \begin{pmatrix}
\chi_1 \eta_1 & \chi_1 \eta_2 & \cdots & \chi_1 \eta_n \\
\chi_2 \eta_1 & \chi_2 \eta_2 & \cdots & \chi_2 \eta_n \\
\vdots & \vdots & \ddots & \vdots \\
\chi_m \eta_1 & \chi_m \eta_2 & \cdots & \chi_m \eta_n
\end{pmatrix}
\]

**Definition 1.11** Let \( x \in \mathbb{R}^m \), \( y \in \mathbb{R}^n \), and \( A \in \mathbb{R}^{m \times n} \). Then a rank-1 update to matrix \( A \) is given by

\[
A \leftarrow xy^T + A
\]

1.3.3 Matrix-matrix operations

(GEMM) Matrix times a matrix

**Definition 1.12** Given matrices \( A \in \mathbb{R}^{m \times k} \) and \( B \in \mathbb{R}^{k \times n} \) the matrix-matrix product is given by the \( m \times n \) matrix

\[
AB = \begin{pmatrix}
\alpha_{11} & \alpha_{12} & \cdots & \alpha_{1k} \\
\alpha_{21} & \alpha_{22} & \cdots & \alpha_{2k} \\
\vdots & \vdots & \ddots & \vdots \\
\alpha_{m1} & \alpha_{m2} & \cdots & \alpha_{mk}
\end{pmatrix} \begin{pmatrix}
\beta_{11} & \beta_{12} & \cdots & \beta_{1n} \\
\beta_{21} & \beta_{22} & \cdots & \beta_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\beta_{k1} & \beta_{k2} & \cdots & \beta_{kn}
\end{pmatrix}
\]

\[
= \begin{pmatrix}
\alpha_{11} \beta_{11} + \cdots + \alpha_{1k} \beta_{k1} & \alpha_{11} \beta_{12} + \cdots + \alpha_{1k} \beta_{k2} & \cdots & \alpha_{11} \beta_{1n} + \cdots + \alpha_{1k} \beta_{kn} \\
\alpha_{21} \beta_{11} + \cdots + \alpha_{2k} \beta_{k1} & \alpha_{21} \beta_{12} + \cdots + \alpha_{2k} \beta_{k2} & \cdots & \alpha_{21} \beta_{1n} + \cdots + \alpha_{2k} \beta_{kn} \\
\vdots & \vdots & \ddots & \vdots \\
\alpha_{m1} \beta_{11} + \cdots + \alpha_{mk} \beta_{k1} & \alpha_{m1} \beta_{12} + \cdots + \alpha_{mk} \beta_{k2} & \cdots & \alpha_{m1} \beta_{1n} + \cdots + \alpha_{mk} \beta_{kn}
\end{pmatrix}
\]
1.3. OPERATIONS ON VECTORS AND MATRICES

Note: If $C = AB$ where $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$, then $\gamma_{ij} = \sum_{p=1}^{k} \alpha_{ip} \beta_{pj}$

Note: A (column) vector of length $n$ can be viewed as a $n \times 1$ matrix. A row vector of length $n$ can be viewed as a $1 \times n$ matrix.

- If $x, y \in \mathbb{R}^n$ then performing matrix-matrix multiplication treating $x$ and $y$ as matrices shows that $x^T y$ equals the inner product of the two vectors.
- If $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$ then performing matrix-matrix multiplication treating $x$ and $y$ as matrices shows that $xy^T$ equals the outer product of the two vectors.
- If $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, then performing matrix-matrix multiplication treating $x$ and $y$ as matrices shows that $y = Ax$ equals the matrix-vector multiplication. In other words, the definition of matrix-matrix multiplication is such that it is consistent with the definition of matrix-vector multiplication when vectors are treated as special cases of matrices.

1.3.4 Partitioning

Frequently, we will find it convenient to partition matrices and vectors into submatrices and subvectors.

Partitioning vectors

By definition, a vector $x \in \mathbb{R}^n$ is the ordered collection of elements

$$x = \begin{pmatrix} 
\chi_1 \\
\chi_2 \\
\vdots \\
\chi_n
\end{pmatrix}$$

We can partition this vector $x$ into subvectors $x_i$, $i = 1, \ldots, N$ like

$$x = \begin{pmatrix} 
x_1 \\
x_2 \\
\vdots \\
x_N
\end{pmatrix} \quad (1.1)$$

where $x_i$ is a vector of length $n_i$. By this partitioning we mean that

$$x_1 = \begin{pmatrix} 
\chi_1 \\
\chi_2 \\
\vdots \\
x_{n_1}
\end{pmatrix}, \quad x_2 = \begin{pmatrix} 
\chi_{n_1+1} \\
\chi_{n_1+2} \\
\vdots \\
x_{n_1+n_2}
\end{pmatrix},$$

and so forth. Clearly, $n = \sum_{i=1}^{N} n_i$.

Note: The case where $n_i = 1$ yields a subvector that is also a scalar while the case where $n_i = 0$ yields a subvector that is empty (has no elements). Both of these special cases are useful in practice.

Note: Let $x \in \mathbb{R}^n$ be partitioned as in (1.1). Then $x^T = (x_1^T | x_2^T | \cdots | x_N^T)$.

Note: Let $x \in \mathbb{R}^n$ be partitioned as in (1.1) and let $y \in \mathbb{R}^n$ be partitioned conformally by which we mean that

$$y = \begin{pmatrix} 
y_1 \\
y_2 \\
\vdots \\
y_N
\end{pmatrix}$$
where \( x_i, y_i \in \mathbb{R}^{n_i} \), \( i = 1, \ldots, N \). Then the inner product of \( x \) and \( y \) is given by

\[
x^T y = \begin{pmatrix} x_1^T & x_2^T & \cdots & x_N^T \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \sum_{i=1}^{N} x_i^T y_i
\]

**Partitioning matrices**

Just like vectors can be partitioned into subvectors, it is often useful to partition matrices into submatrices. Let \( A \in \mathbb{R}^{m \times n} \). This matrix can be partitioned into a \( M \times N \) partitioned matrix like

\[
A = \begin{pmatrix}
A_{11} & A_{12} & \cdots & A_{1N} \\
A_{21} & A_{22} & \cdots & A_{2N} \\
\vdots & \vdots & \ddots & \vdots \\
A_{M1} & A_{M2} & \cdots & A_{MN}
\end{pmatrix}
\]

where \( A_{ij} \in \mathbb{R}^{m_i \times n_j} \) with \( \sum_{i=1}^{M} m_i = m \) and \( \sum_{j=1}^{N} n_j = n \). Here

\[
A_{11} = \begin{pmatrix}
\alpha_{11} & \cdots & \alpha_{1n_1} \\
\vdots & \ddots & \vdots \\
\alpha_{m_1 n_1} & \cdots & \alpha_{m_1 n_1}
\end{pmatrix}, \quad
A_{12} = \begin{pmatrix}
\alpha_{1(n_1+1)} & \cdots & \alpha_{1(n_1+n_2)} \\
\vdots & \ddots & \vdots \\
\alpha_{m_1(n_1+n_2)} & \cdots & \alpha_{m_1(n_1+n_2)}
\end{pmatrix}, \quad
A_{21} = \begin{pmatrix}
\vdots \\
\alpha_{(m_1+1) n_1} & \cdots & \alpha_{(m_1+1) n_1}
\end{pmatrix}, \quad
A_{22} = \begin{pmatrix}
\alpha_{(m_1+1)(n_1+1)} & \cdots & \alpha_{(m_1+1)(n_1+n_2)} \\
\vdots & \ddots & \vdots \\
\alpha_{(m_1+1)(n_1+n_2)} & \cdots & \alpha_{(m_1+1)(n_1+n_2)}
\end{pmatrix}, \quad
\vdots
\]

**Theorem 1.1** Let

\[
C = \begin{pmatrix}
C_{11} & \cdots & C_{1N} \\
\vdots & \ddots & \vdots \\
C_{M1} & \cdots & C_{MN}
\end{pmatrix}, \quad
A = \begin{pmatrix}
A_{11} & \cdots & A_{1K} \\
\vdots & \ddots & \vdots \\
A_{M1} & \cdots & A_{MK}
\end{pmatrix}, \quad
B = \begin{pmatrix}
B_{11} & \cdots & B_{1N} \\
\vdots & \ddots & \vdots \\
B_{K1} & \cdots & B_{KN}
\end{pmatrix},
\]

where \( C_{ij} \in \mathbb{R}^{m_i \times n_j} \), \( A_{ip} \in \mathbb{R}^{m_i \times k_p} \), and \( B_{pj} \in \mathbb{R}^{k_p \times n_j} \), with \( \sum_{i=1}^{M} m_i = m, \sum_{j=1}^{N} n_j = n \) and \( \sum_{p} k_p = k \). Then

\[
C_{ij} = \sum_{l=1}^{K} A_{il} B_{lj}
\]

**Proof:** We will prove the above only for the case where \( C, A, \) and \( B \) are partitioned into \( 2 \times 2 \) blocked matrices: Let \( A, B, \) and \( C \) all be partitioned so that

\[
C = \begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix}, \quad
A = \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}, \quad
B = \begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}
\]

with

\[
C_{11} \in \mathbb{R}^{m_1 \times n_1}, \quad A_{11} \in \mathbb{R}^{m_1 \times k_1}, \quad C_{11} \in \mathbb{R}^{k_1 \times n_1}, \quad C_{21} \in \mathbb{R}^{m_2 \times n_1}, \quad A_{21} \in \mathbb{R}^{m_2 \times k_1}, \quad C_{21} \in \mathbb{R}^{k_2 \times n_1}, \quad C_{12} \in \mathbb{R}^{m_1 \times n_2}, \quad A_{12} \in \mathbb{R}^{m_1 \times k_2}, \quad C_{12} \in \mathbb{R}^{k_1 \times n_2}, \quad C_{22} \in \mathbb{R}^{m_2 \times n_2}, \quad A_{22} \in \mathbb{R}^{m_2 \times k_2}, \quad C_{22} \in \mathbb{R}^{k_2 \times n_2}
\]
with \(m_1 + m_2 = m\), \(n_1 + n_2 = n\), and \(k_1 + k_2 = k\). Then

\[
C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}
\]

rest of proof here...

\[\square\]

Example 1.1 Let

\[
A = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 1 & -1 \\ 0 & 1 & 2 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1 & -2 & 1 & 0 \\ 0 & -1 & 1 & 2 \\ -1 & 1 & 2 & 1 \end{pmatrix}
\]

Then

\[
C = AB = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 1 & -1 \\ 0 & 1 & 2 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -2 & 1 & 0 \\ 0 & -1 & 1 & 2 \\ -1 & 1 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & -2 \\ 3 & -6 & 1 & 1 \\ -2 & 1 & 5 & 4 \\ 0 & 0 & 2 & -1 \end{pmatrix}
\]

Partitioning

\[
C = \begin{pmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \\ \gamma_{31} & \gamma_{32} \\ \gamma_{41} & \gamma_{42} \end{pmatrix} \begin{pmatrix} \gamma_{13} & \gamma_{14} \\ \gamma_{23} & \gamma_{24} \\ \gamma_{33} & \gamma_{34} \\ \gamma_{43} & \gamma_{44} \end{pmatrix}, \quad A = \begin{pmatrix} 1 & -1 \\ 2 & 1 \\ 0 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \end{pmatrix}, \quad \text{and} \quad B = \begin{pmatrix} 1 & -2 \\ 0 & -1 \\ 1 & 2 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}
\]

we see that

\[
\begin{pmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \\ \gamma_{31} & \gamma_{32} \\ \gamma_{41} & \gamma_{42} \end{pmatrix} \begin{pmatrix} \gamma_{13} & \gamma_{14} \\ \gamma_{23} & \gamma_{24} \\ \gamma_{33} & \gamma_{34} \\ \gamma_{43} & \gamma_{44} \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 2 & 1 \\ 0 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 & -2 \\ 0 & -1 \\ 1 & 2 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}
\]
Definition 1.13 Let \( x \in \mathbb{R}^n, y \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n} \) and consider \( y = Ax \). Let \( x \) and \( y \) be partitioned like
\[
\begin{pmatrix}
  x_1 \\
  \vdots \\
  x_N \\
\end{pmatrix}
\quad \text{and} \quad
\begin{pmatrix}
  y_1 \\
  \vdots \\
  y_M \\
\end{pmatrix}
\]
(1.2)
where \( x_j \in \mathbb{R}^{n_j} \) and \( y_i \in \mathbb{R}^{m_i} \), with \( \sum_{j}^{N} x_j = n, \sum_{i}^{M} y_i = m \). Then \( A \) is said to be partitioned conformally with \( x \) and \( y \), if
\[
A = \begin{pmatrix}
  A_{11} & \cdots & A_{1N} \\
  \vdots & \ddots & \vdots \\
  A_{M1} & \cdots & A_{MN} \\
\end{pmatrix}
\]
where \( A_{ij} \in \mathbb{R}^{m_i \times n_j} \).

The reason why a conformal partitioning of \( A \) is desirable becomes clear when considering the following corollary:

Corollary 1.14 If \( x \) and \( y \) are partitioned as in (1.2) and \( A \) is partitioned conformally with respect to \( x \) and \( y \), then \( y = Ax \) implies that
\[
\begin{pmatrix}
  y_1 \\
  \vdots \\
  y_M \\
\end{pmatrix}
= \begin{pmatrix}
  A_{11} & \cdots & A_{1N} \\
  \vdots & \ddots & \vdots \\
  A_{M1} & \cdots & A_{MN} \\
\end{pmatrix}
\begin{pmatrix}
  x_1 \\
  \vdots \\
  x_N \\
\end{pmatrix}
= \begin{pmatrix}
  A_{11}x_1 + \cdots + A_{1N}x_N \\
  \vdots \\
  A_{M1}x_1 + \cdots + A_{MN}x_N \\
\end{pmatrix}
\]
In other words,
\[
y_i = A_{i1}x_1 + \cdots + A_{iN}x_N
\]
Notice that the matrix-vector multiplications in each term on the right only make sense if \( A_{ij} \) has the same number of columns as \( x_j \) has elements. Also, the results of these terms must have \( m_i \) elements so that the terms can be added together to yield \( y_i \).

Example 1.2 Let
\[
A = \begin{pmatrix}
  1 & -1 & 0 & 2 \\
  2 & 1 & -1 & -2 \\
  0 & 1 & 2 & -1 \\
  1 & -1 & 1 & 0 \\
\end{pmatrix}
\quad \text{and} \quad
x = \begin{pmatrix}
  1 \\
  -1 \\
  0 \\
  2 \\
\end{pmatrix}
\]
Then
\[
y = Ax = \begin{pmatrix}
  1 & -1 & 0 & 2 \\
  2 & 1 & -1 & -2 \\
  0 & 1 & 2 & -1 \\
  1 & -1 & 1 & 0 \\
\end{pmatrix}
\begin{pmatrix}
  1 \\
  -1 \\
  0 \\
  2 \\
\end{pmatrix}
= \begin{pmatrix}
  1 \times 1 - 1 \times -1 + 0 \times 0 + 2 \times 2 \\
  2 \times 1 + 1 \times -1 - 1 \times 0 - 2 \times 2 \\
  0 \times 1 + 1 \times -1 + 2 \times 0 - 1 \times 2 \\
  1 \times 1 - 1 \times -1 + 1 \times 0 + 0 \times 2 \\
\end{pmatrix}
= \begin{pmatrix}
  6 \\
  -3 \\
  -3 \\
  2 \\
\end{pmatrix}
\]
Partitioning
\[
x = \begin{pmatrix}
  x_1 \\
  x_2 \\
\end{pmatrix}
, \quad
y = \begin{pmatrix}
  y_1 \\
  y_2 \\
\end{pmatrix}
, \quad \text{and} \quad
A = \begin{pmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22} \\
\end{pmatrix}
\]
where \( A_{11}, A_{12}, A_{21}, \) and \( A_{22} \) are \( 2 \times 2 \) and \( x_1, x_2, y_1, \) and \( y_2 \) have 2 elements each, we see that
\[
\begin{pmatrix}
  \eta_1 \\
  \eta_2 \\
  \eta_3 \\
  \eta_4 \\
\end{pmatrix}
= \begin{pmatrix}
  \begin{pmatrix}
  1 & -1 \\
  0 & 1 \\
  1 & -1 \\
\end{pmatrix}
  & \begin{pmatrix}
  0 & 2 \\
  2 & 1 \\
  1 & 0 \\
\end{pmatrix}
  & \begin{pmatrix}
  1 & 0 \\
  -1 & 1 \\
  -1 & 1 \\
\end{pmatrix}
  & \begin{pmatrix}
  0 & 2 \\
  -1 & -1 \\
  0 & 2 \\
\end{pmatrix}
\end{pmatrix}
\begin{pmatrix}
  \begin{pmatrix}
  1 \\
  0 \\
\end{pmatrix}
  & \begin{pmatrix}
  -1 \\
  1 \\
  1 \\
\end{pmatrix}
  & \begin{pmatrix}
  0 \\
  2 \\
  0 \\
\end{pmatrix}
  & \begin{pmatrix}
  0 \\
  -1 \\
  0 \\
\end{pmatrix}
\end{pmatrix}
\]
\[
= \begin{pmatrix}
  \begin{pmatrix}
  1 & -1 \\
  2 & 1 \\
  1 & -1 \\
\end{pmatrix}
  & \begin{pmatrix}
  1 \\
  -1 \\
  1 \\
\end{pmatrix}
  & \begin{pmatrix}
  0 & 2 \\
  -1 & -1 \\
  0 & 2 \\
\end{pmatrix}
  & \begin{pmatrix}
  0 \\
  -1 \\
  0 \\
\end{pmatrix}
\end{pmatrix}
\]
\[\begin{pmatrix} 2 \\ 1 \\ -1 \\ 2 \end{pmatrix} + \begin{pmatrix} 4 \\ -4 \\ -2 \\ 0 \end{pmatrix} = \begin{pmatrix} 6 \\ -3 \\ -3 \\ 2 \end{pmatrix}\]

**Theorem 1.2** If

\[A = \begin{pmatrix} A_{11} & \cdots & A_{1K} \\ \vdots & & \vdots \\ A_{M1} & \cdots & A_{MK} \end{pmatrix}\]

then

\[A^T = \begin{pmatrix} A_{11} & \cdots & A_{1K} \\ \vdots & & \vdots \\ A_{M1} & \cdots & A_{MK} \end{pmatrix}^T \]

\[= \begin{pmatrix} A_{11}^T & \cdots & A_{K1}^T \\ \vdots & & \vdots \\ A_{1M}^T & \cdots & A_{KM}^T \end{pmatrix} \]
Chapter 2

Practicalities

We will assume all numbers involved are double-precision real.

2.1 Storage

2.1.1 Matrices

Let us assume that we wish to store \( m \times n \) matrix \( A \), of data-type double-precision real. Since we will often interface with FORTRAN, we will use column-major storage. Using dynamic data allocation, in C we may achieve this with the following code segment:

```c
int m, n;
double *a;
...
a = (double *) malloc( m * n * sizeof( double ) );
```

Element \( a_{ij} \) can now be stored in array element \( a[ (j-1)\times m + (i-1) ] \). In other words, all elements of the first column are stored first, followed by the second column, and so forth.

Frequently, we will need an array large enough to hold a \( M \times N \) matrix, of which we will only use the upper-left \( m \times n \) submatrix. In this case, we may use the code segment

```c
int M, N, m, n;
double *a;
...
a = (double *) malloc( M * N * sizeof( double ) );
```

and element \( a_{ij} \) is now stored in \( a[ (j-1)\times M + (i-1) ] \). It is important to note that while the matrix being stored is \( m \times n \), it is the row dimension of the \( M \times N \) array that determines how the elements of this smaller matrix are stored. Notice that when computing where in memory \( a_{ij} \) exists, the column dimension \( N \) is irrelevant. Dimension \( M \) of the larger array is typically referred to as the leading dimension of the matrix.

To fully understand the concept of a leading dimension, consider the example in Figure 2.1, In that figure, we have a main driver that queries the biggest matrix that will may encountered, and creates an array to hold it. Next, it queries the actual size of a matrix to be created and calls a subroutine to fill it. The catch is that the matrix to be created is to occupy the upper-left corner of the original array. If the leading dimension isn’t passed to the subroutine, that subroutine has no knowledge of how the matrix is actually stored.
main() 
{ 
    double *a;
    int lda, n_max, m, n;
    
    printf("enter dimensions of largest matrix that may be encountered:n");
    scanf("%d%d", &lda, &n_max);
    a = (double *)malloc(lda * n_max * sizeof(double));
    
    printf("enter dimensions of current matrix to be used:");
    scanf("%d%d", &m, &n);
    
    /* call subroutine to fill the m x n matrix */
    matrix_fill(m, n, a, lda);
    
    ...
}

#define B(i, j, ldb) (b[((j)-1)*(ldb) + (i)-1])

void matrix_fill(int mm, int nn, double *b, int ldb) 
{ 
    int i, j;
    
    for (j=1; j<=nn; j++) { 
        for (i=1; i<=mm; i++) { 
            B(i, j, ldb) = <value of i,j entry of matrix>;
        }
    }
}

Figure 2.1: Example of use of leading dimension.
2.1.2 Vectors

Let us assume that we wish to store a vector $x$ of length $n$, with entries

$$x = \begin{pmatrix}
    x_1 \\
    x_2 \\
    \vdots \\
    x_n
\end{pmatrix}$$

Notice that the following code segment could be used to create space to store $x$:

```c
int n
double *x;
...
x = (double *) malloc( n * sizeof( double ) );
```

Now $x_i$ can be stored in $x[ i-1 ]$.

But vectors are encountered in many forms. For example, assume we have $m \times n$ matrix $A$ stored in array $a$ with leading dimension $lda$. Now assume we wish creating a row or a column of this matrix, viewing it as a vector. This is illustrated in Figure 2.2. Notice that the address of where the first element of a vector is stored together with the increment indicating how to step through memory to get from one element of the vector to next totally describes how to access the elements of the vector.

2.2 Implementing Matrix-Vector Multiplication

Let us examine how to implement $y = \alpha A x + \beta y$ where $A$ is an $m \times n$ matrix. Notice that we can compute

$$y = \alpha A x + \beta y = \alpha \begin{pmatrix} a_1 & a_2 & \cdots & a_n \end{pmatrix} \begin{pmatrix} x_1 \\
    x_2 \\
    \vdots \\
    x_n
\end{pmatrix} + \beta y$$

which translates to the code

```c
#define X( i, incx ) x[ (((i)-1)*(incx) ]
#define Y( i, incy ) y[ (((i)-1)*(incy) ]
#define A( i, j, lda ) a[ (((j)-1)*(lda)+(i)-1 ]

void mmult_axy( int m, int n, double alpha, double *a, int lda,
                double *x, int incx, double beta, double *y, int incy )
{
    int i, j;
    double temp;

    for ( i=1; i<=m; i++ ) Y( i, incy ) = beta * Y( i, incy );

    for ( j=1; j<=n; j++ ) {
        temp = alpha * X( j, incx );
        for ( i=1; i<=m; i++ )
            Y( i, incy ) += A( i, j, lda ) * temp;
    }
}
```
#define A( i, j, lda ) a[ ((j)-1)*(lda) + (i)-1 ]
main()
{
    double *a;
    int lda, n_max, m, n;

    printf("Enter dimensions of largest matrix that may be encountered\n:");
    scanf( "%d%d", &lda, &n_max );

    a = ( double * ) malloc( lda * n_max * sizeof( double ) );

    printf("Enter dimensions of current matrix to be used:");
    scanf( "%d%d", &m, &n );

    /* call subroutine to fill the m x n matrix */
    matrix_fill( m, n, a, lda );

    /* print the 5th column of the matrix */
    print_vector( m, &A( 1, 5, lda ), 1 );

    /* print the 3rd row of the matrix */
    print_vector( n, &A( 3, 1, lda ), lda );
}

#define X( i, incx ) ( x[ ((i)-1)*incx ] )

void print_vector ( int n, double *x, int incx )
{
    int i;

    for ( i=1; i<=n; i++ ) {
        printf("x%d = %lf\n", i, X( i, incx ) );
    }
}

Figure 2.2: Example of use of increment
The operation \( y \leftarrow ax + y \) is provided by the Basic Linear Algebra Subprograms (BLAS) routine `cblas_daxpy`:

```c
void cblas_daxpy( int n, double alpha, double *x, int incx, double *y, int incy );
```

Here

- \( n \) equals the length of the vectors \( x \) and \( y \),
- \( a \) is stored in `alpha`,
- \( x \) is stored in memory at address \( x \), striding through memory as given in `incx`, and
- \( y \) is stored in memory at address \( y \), striding through memory as given in `incy`.

Thus, the above matrix-vector multiplication can be rewritten like

```c
#include "cblas.h"

void mvmult_daxpy( int m, int n, double alpha, double *a, int lda,
                    double *x, int incx, double beta, double *y, int incy )
{
   int i, j;
   double temp;

   for ( i=1; i<=m; i++ ) Y( i, incy ) = beta * Y( i, incy );

   for ( j=1; j<=n; j++ ) {
      temp = alpha * X( j, incx );
      cblas_daxpy( m, temp, &A( i, j, lda ), 1, y, incy );
   }
}
```

Alternatively, we may wish to use the formula

\[
y = \begin{pmatrix}
\eta_1 \\
\vdots \\
\eta_n
\end{pmatrix} = \alpha Ax + \beta y = \alpha \begin{pmatrix}
a_1^T \\
\vdots \\
a_m^T
\end{pmatrix} x + \beta \begin{pmatrix}
\eta_1 \\
\vdots \\
\eta_n
\end{pmatrix} = \begin{pmatrix}
\alpha a_1^T x + \beta \eta_1 \\
\vdots \\
\alpha a_m^T x + \beta \eta_m
\end{pmatrix}
\]

which can be coded by restructuring the loops:

```c
#define X( i, incx ) x[ ((i)-1)*(incx) ]
#define Y( i, incy ) y[ ((i)-1)*(incy) ]
#define A( i, j, lda ) a[ ((j)-1)*(lda)+(i)-1 ]

void mvmult_dot( int m, int n, double alpha, double *a, int lda,
                 double *x, int incx, double beta, double *y, int incy )
{
   int i, j;
   double temp;

   for ( i=1; i<=m; i++ ) {
      temp = 0.0;
      for ( j=1; j<=n; j++ )
         Y( i, incy ) = Y( i, incy ) + A( i, j, lda ) * X( j );
      Y( i, incy ) = alpha * temp + beta * Y( i, incy );
   }
}
```
The operation $\alpha \leftarrow x^T y$ is provided by the BLAS routine `cblas_ddot`:

```c
double cblas_ddot( int n, double *x, int incx, double *y, int incy );
```

Here

- $n$ equals the length of the vectors $x$ and $y$,
- $x$ is stored in memory at address $x$, striding through memory as given in $incx$, and
- $y$ is stored in memory at address $y$, striding through memory as given in $incy$.

The routine returns the value $x^T y$.

Thus, the above dot-product based implementation of matrix-vector multiplication can be rewritten like:

```c
void mvmult_dot( int m, int n, double alpha, double *a, int lda,
                 double *x, int incx, double beta, double *y, int incy )
{
  int i, j;

  for ( i=1; i<=m; i++ )
    Y( i, incy ) =
      alpha * cblas_ddot( n, &A( i, 1, lda ), lda, x, incx ) +
      beta * Y( i, incy );
} 
```