

An Introduction to Linear Algebra

Barry M. Wise and Neal B. Gallagher
Eigenvector Research, Inc.
830 Wapato Lake Road
Manson, WA 98831
USA
bmw@eigenvector.com

Linear algebra is the language of chemometrics. One cannot expect to truly understand most chemometric techniques without a basic understanding of linear algebra. This article reviews the basics of linear algebra and provides the reader with the foundation required for understanding most chemometrics literature. It is presented in a rather dense fashion: no proofs are given and there is little discussion of the theoretical implications of the theorems and results presented. The goal has been to condense into as few pages as possible the aspects of linear algebra used in most chemometric methods. Readers who are somewhat familiar with linear algebra may find this article to be a good quick review. Those totally unfamiliar with linear algebra should consider spending some time with a linear algebra text. In particular, those by Gilbert Strang are particularly easy to read and understand. Several of the numerical examples in this section are adapted from Strang's *Linear Algebra and Its Applications, Second Edition* (Academic Press, 1980).

MATLAB (The MathWorks, Inc., Natick MA) commands for performing the operations listed are also included; the reader is encouraged to run the examples presented in the text. Those unfamiliar with MATLAB may wish to read the first few sections of the tutorial chapter of the MATLAB User's Guide.

Scalars, Vectors and Matrices

A *scalar* is a mathematical quantity that is completely described by a magnitude, *i.e.* a single number. Scalar variables are generally denoted by lowercase letters, *e.g.* a . Examples of scalar variables include temperature, density, pressure and flow. In MATLAB, a value can be assigned to a scalar at the command line, *e.g.*

```
»a = 5;
```

Here we have used the semicolon operator to suppress the echo of the result. Without this semicolon MATLAB would display the result of the assignment:

```
»a = 5
```

```
a =
```

```
5
```

A *vector* is a mathematical quantity that is completely described by its magnitude and direction. An example of a three dimensional column vector might be

$$\mathbf{b} = \begin{bmatrix} 4 \\ 3 \\ 5 \end{bmatrix} \quad (1)$$

Vectors are generally denoted by bold lowercase letters. (In MATLAB no distinction is made between the notation for scalars, vectors and matrices: they can be upper or lower case and bold letters are not used.) In MATLAB, this vector could be entered at the command in one of several ways. One way would be to enter the vector with an element on each line, like this:

```
»b = [4
3
5]
```

b =

```
4
3
5
```

Another way to enter the vector would be to use the semicolon to tell MATLAB that each line was completed. For instance

```
»b = [4; 3; 5];
```

produces the same result as above.

This vector is represented geometrically in Figure 1, where the three components 4, 3 and 5 are the coordinates of a point in three-dimensional space. Any vector **b** can be represented by a point in space; there is a perfect match between points and vectors. One can choose to think of a vector as the arrow, the point in space, or as the three numbers which describe the point. In a problem with 400 dimensions (such as in spectroscopy), it is probably easiest to consider the 400 numbers. The transpose of a column vector is a row vector and vice-versa. The transpose is generally indicated by a superscript T, *i.e.* \mathbf{b}^T , though in some instances, including MATLAB, an apostrophe (') will be used. For example

$$\mathbf{b}^T = [4 \quad 3 \quad 5] \quad (2)$$

In MATLAB, we could easily assign \mathbf{b}^T to another variable **c**, as follows:

```
»c = b'
```

c =

```
4           3           5
```

A *matrix* is a rectangular array of scalars, or in some instances, algebraic expressions which evaluate to scalars. Matrices are said to be *m* by *n*, where *m* is the number of rows in the matrix and *n* is the number of columns. A 3 by 4 matrix is shown here

$$\mathbf{A} = \begin{bmatrix} 2 & 5 & 3 & 6 \\ 7 & 3 & 2 & 1 \\ 5 & 2 & 0 & 3 \end{bmatrix} \quad (3)$$

This matrix could be entered in MATLAB as follows:

```
»A = [2 5 3 6; 7 3 2 1; 5 2 0 3];
```

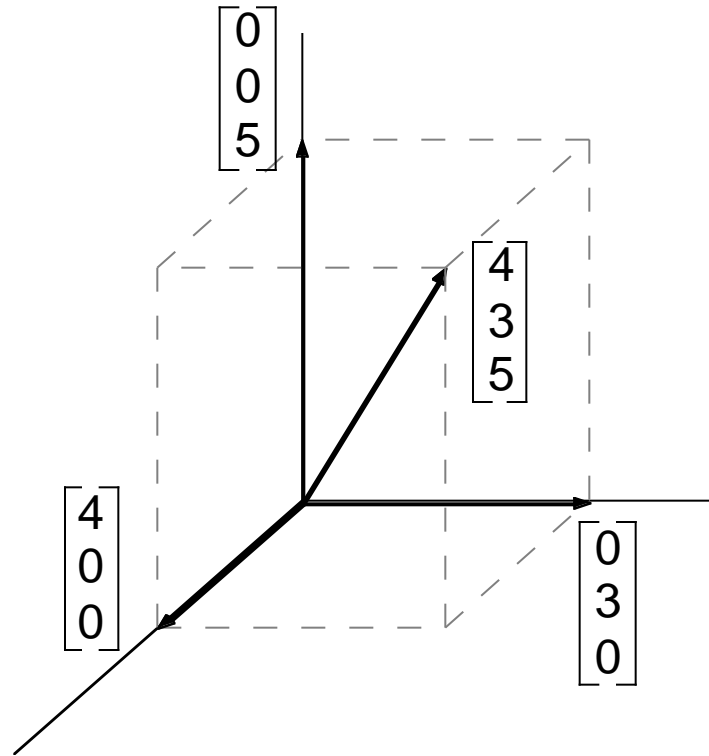


Figure 1. Geometric Representation of the Vector $[4 \ 3 \ 5]^T$

Matrices are usually denoted by bold uppercase letters. The elements of a matrix can be indicated by their row and column indices, for instance, $A_{2,4} = 1$. We can index individual matrix elements in MATLAB in a similar way, for instance:

```
»A(2,4)
```

```
ans =
```

```
1
```

The transpose operator “flips” a matrix along its diagonal elements, creating a new matrix with the i^{th} row being equal to the j^{th} column of the original matrix, *e.g.*

$$\mathbf{A}^T = \begin{bmatrix} 2 & 7 & 5 \\ 5 & 3 & 2 \\ 3 & 2 & 0 \\ 6 & 1 & 3 \end{bmatrix} \quad (4)$$

This would be done in MATLAB:

```
»A'
```

```
ans =
```

```
2      7      5
5      3      2
3      2      0
6      1      3
```

Vectors and scalars are special cases of matrices. A vector is a matrix with either one row or column. A scalar is a matrix with a single row and column.

Vector and Matrix Addition and Multiplication

Matrices can be added together provided that the dimensions are consistent, *i.e.* both matrices must have the same number of rows and columns. To add two matrices together simply add the corresponding elements. For instance:

$$\begin{bmatrix} 1 & 4 & 3 \\ 5 & 4 & 0 \end{bmatrix} + \begin{bmatrix} 2 & 4 & 1 \\ 2 & 6 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 8 & 4 \\ 7 & 10 & 3 \end{bmatrix} \quad (5)$$

In MATLAB we might do this as follows:

```
»x = [1 4 3; 5 4 0]; y = [2 4 1; 2 6 3];
```

```
»x + y
```

```
ans =
```

```
3      8      4
7     10      3
```

Note that if the matrices are not of consistent sizes, the addition will not be defined:

$$\begin{bmatrix} 1 & 4 & 3 \\ 5 & 4 & 0 \end{bmatrix} + \begin{bmatrix} 2 & 4 \\ 1 & 2 \\ 6 & 3 \end{bmatrix} = ?? \quad (6)$$

If you try this in MATLAB, you will get the following:

```
»x = [1 4 3; 5 4 0]; y = [2 4; 1 2; 6 3];
```

```
»x + y
```

```
??? Error using ==> +
```

```
Matrix dimensions must agree.
```

This is one of the most common error messages you will see when using MATLAB. It tells you that the operation you are attempting is not defined.

Vector and matrix addition is commutative and associative, so provided that **A**, **B** and **C** are matrices with the same number of rows and columns, then:

$$\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A} \quad (7)$$

$$(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C}) \quad (8)$$

Vectors and matrices can be multiplied by a scalar. For instance, if $c = 2$, then (using our previously defined **A**):

$$c\mathbf{A} = \begin{bmatrix} 4 & 10 & 6 & 12 \\ 14 & 6 & 4 & 2 \\ 10 & 4 & 0 & 6 \end{bmatrix}$$

Multiplication of a matrix by a scalar in MATLAB is done using the `*` operator:

```
>>c = 2;
```

```
>>c*A
```

```
ans =
```

```

      4      10      6      12
     14      6      4      2
     10      4      0      6
```

Note that scalar multiplication is commutative and associative, so if c and d are scalars:

$$c\mathbf{A} = \mathbf{A}c \quad (9)$$

$$(c + d)\mathbf{A} = c\mathbf{A} + d\mathbf{A} \quad (10)$$

Vectors can be multiplied together in two different ways. The most common vector multiplication is the *inner product*. In order for the inner product to be defined, the two vectors must have the same number of elements or *dimension*. The inner product is the sum over the products of corresponding elements. For instance, for two 3 element column vectors **a** and **b** with

$$\mathbf{a} = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 4 \\ 3 \\ 5 \end{bmatrix} \quad (11)$$

the inner product is

$$\mathbf{a}^T \mathbf{b} = [2 \quad 5 \quad 1] \begin{bmatrix} 4 \\ 3 \\ 5 \end{bmatrix} = [2*4 + 5*3 + 1*5] = 28 \quad (12)$$

Note that the inner product of two vectors produces a scalar result. The inner product occurs often in the physical sciences and is often referred to as the *dot product*. We will see it again in regression problems where we model a system property or output (such as a chemical concentration or quality variable) as the weighted sum of a number of different measurements. In MATLAB, this example looks like:

```
»a = [2; 5; 1]; b = [4; 3; 5];
»a'*b
```

```
ans =
```

```
28
```

The inner product is also used when calculating the length of a vector. The length of a vector is square root of the sum of squares of the vector coefficients. The length of a vector **a**, denoted $\|\mathbf{a}\|$ (also known as the 2-norm of the vector), is the square root of the inner product of the vector with itself:

$$\|\mathbf{a}\| = \sqrt{\mathbf{a}^T \mathbf{a}} \quad (13)$$

We could calculate the norm of the vector **a** above explicitly in MATLAB with the `sqrt` (square root) function. Alternately, we could use the `norm` function. Both methods are shown here:

```
»sqrt(a'*a)
```

```
ans =
```

```
5.4772
```

```
»norm(a)
```

```
ans =
```

```
5.4772
```

The vector *outer product* is encountered less often than the inner product, but will be important in many instances in chemometrics. Any two vectors of arbitrary dimension (number of elements) can be multiplied together in an outer product. The result will be an m by n matrix where m is the number of elements in the first vector and n is the number of elements in the second vector. As an example, take

$$\mathbf{a} = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 4 \\ 3 \\ 5 \\ 7 \\ 9 \end{bmatrix} \quad (14)$$

The outer product of **a** and **b** is then

$$\mathbf{ab}^T = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix} \begin{bmatrix} 4 & 3 & 5 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 2*4 & 2*3 & 2*5 & 2*7 & 2*9 \\ 5*4 & 5*3 & 5*5 & 5*7 & 5*9 \\ 1*4 & 1*3 & 1*5 & 1*7 & 1*9 \end{bmatrix}$$

$$= \begin{bmatrix} 8 & 6 & 10 & 14 & 18 \\ 20 & 15 & 25 & 35 & 45 \\ 4 & 3 & 5 & 7 & 9 \end{bmatrix} \quad (15)$$

In MATLAB this would be:

```
»a = [2 5 1]'; b = [4 3 5 7 9]';
»a*b'
```

ans =

```
      8      6     10     14     18
     20     15     25     35     45
      4      3      5      7      9
```

Here we used a slightly different method of entering a column vector: it was entered as the transpose of a row vector.

Orthogonal and Orthonormal Vectors

Vectors are said to be *orthogonal* if their inner product is zero. The geometric interpretation of this is that they are at right angles to each other or perpendicular. Vectors are *orthonormal* if they are orthogonal and of unit length, *i.e.* if their inner product with themselves is unity. For an orthonormal set of column vectors \mathbf{v}_i , with $i = 1, 2, \dots, n$, then

$$\mathbf{v}_i^T \mathbf{v}_j = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases} \quad (16)$$

Note that it is impossible to have more than n orthonormal vectors in an n -dimensional space. For instance, in three dimensions, one can only have 3 vectors which are orthogonal, and thus, only three vectors can be orthonormal (although there are an infinite number of sets of 3 orthogonal vectors). Sets of orthonormal vectors can be thought of as new basis vectors for the space in which they are contained. Our conventional coordinate system in 3 dimensions consisting of the vectors $[1 \ 0 \ 0]^T$, $[0 \ 1 \ 0]^T$ and $[0 \ 0 \ 1]^T$ is the most common orthonormal basis set, however, as we shall see, it is not the only basis set, nor is it always the most convenient for describing real data.

Two matrices \mathbf{A} and \mathbf{B} can be multiplied together provided that the number of columns in \mathbf{A} is equal to the number of rows in \mathbf{B} . The result will be a matrix \mathbf{C} which has as many rows as \mathbf{A} and columns as \mathbf{B} . Thus, one can multiply $\mathbf{A}_{m \times n}$ by $\mathbf{B}_{n \times p}$ and the result will be $\mathbf{C}_{m \times p}$. The ij^{th} element of \mathbf{C} consists of the inner product of the i^{th} row of \mathbf{A} with the j^{th} column of \mathbf{B} . As an example, if

$$\mathbf{A} = \begin{bmatrix} 2 & 5 & 1 \\ 4 & 5 & 3 \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} 4 & 3 & 5 & 7 \\ 9 & 5 & 3 & 4 \\ 5 & 3 & 6 & 7 \end{bmatrix} \quad (17)$$

then

$$\mathbf{AB} = \begin{bmatrix} 2*4+5*9+1*5 & 2*3+5*5+1*3 & 2*5+5*3+1*6 & 2*7+5*4+1*7 \\ 4*4+5*9+3*5 & 4*3+5*5+3*3 & 4*5+5*3+3*6 & 4*7+5*4+3*7 \end{bmatrix}$$

$$= \begin{bmatrix} 58 & 34 & 31 & 41 \\ 76 & 46 & 53 & 69 \end{bmatrix} \quad (18)$$

In MATLAB:

```
»A = [2 5 1; 4 5 3]; B = [4 3 5 7; 9 5 3 4; 5 3 6 7];
»A*B
```

```
ans =
```

```
    58    34    31    41
    76    46    53    69
```

Matrix multiplication is distributive and associative

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC} \quad (19)$$

$$(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC}) \quad (20)$$

but is not, in general, commutative

$$\mathbf{AB} \neq \mathbf{BA} \quad (21)$$

Other useful matrix identities include the following

$$(\mathbf{A}^T)^T = \mathbf{A} \quad (22)$$

$$(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T \quad (23)$$

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T \quad (24)$$

Special Matrices

There are several special matrices which deserve some attention. The first is the *diagonal matrix*, which is a matrix whose only non-zero elements lie along the matrix diagonal, *i.e.* the elements for which the row and column indices are equal. An example diagonal matrix **D** is shown here

$$\mathbf{D} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 7 & 0 \end{bmatrix} \quad (25)$$

Diagonal matrices need not have the same number of rows as columns, the only requirement is that only the diagonal elements be non-zero. We shall see that diagonal matrices have additional properties which we can exploit.

Another special matrix of interest is the *identity matrix*, typically denoted as **I**. The identity matrix is always square (number of rows equals number of columns) and contains ones on the diagonal and zeros everywhere else. A 4 by 4 identity matrix is shown here

$$\mathbf{I}_{4 \times 4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (26)$$

Identity matrices can be created easily in MATLAB using the `eye` command. Here we create the 3 by 3 identity matrix:

```
»id = eye(3)
```

```
id =
```

```

1     0     0
0     1     0
0     0     1
```

Any matrix multiplied by the identify matrix returns the original matrix (provided, of course, that the multiplication is defined). Thus, for a matrix $\mathbf{A}_{m \times n}$ and identity matrix \mathbf{I} of appropriate size

$$\mathbf{A}_{m \times n} \mathbf{I}_{n \times n} = \mathbf{I}_{m \times m} \mathbf{A}_{m \times n} = \mathbf{A}_{m \times n} \quad (27)$$

The reader is encouraged to try this with some matrices in MATLAB.

A third type of special matrix is the symmetric matrix. Symmetric matrices must be square and have the property that they are equal to their transpose:

$$\mathbf{A} = \mathbf{A}^T \quad (28)$$

Another way to say this is that $\mathbf{A}_{ij} = \mathbf{A}_{ji}$. We will see symmetric matrices turn up in many places in chemometrics.

Gaussian Elimination: Solving Systems of Equations

Systems of equations arise in many scientific and engineering applications. Linear algebra was largely developed to solve these important problems. Gaussian elimination is the method which is consistently used to solve systems of equations. Consider the following system of 3 equations in 3 unknowns

$$\begin{array}{rrrrrrcl} 2b_1 & + & b_2 & + & b_3 & = & 1 \\ 4b_1 & + & 3b_2 & + & 4b_3 & = & 2 \\ -4b_1 & + & 2b_2 & + & 2b_3 & = & -6 \end{array} \quad (29)$$

This system can also be written using matrix multiplication as follows:

$$\begin{bmatrix} 2 & 1 & 1 \\ 4 & 3 & 4 \\ -4 & 2 & 2 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ -6 \end{bmatrix} \quad (30)$$

or simply as

$$\mathbf{Xb} = \mathbf{y} \quad (31)$$

where

$$\mathbf{X} = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 3 & 4 \\ -4 & 2 & 2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}, \text{ and } \mathbf{y} = \begin{bmatrix} 1 \\ 2 \\ -6 \end{bmatrix} \quad (32)$$

The problem is to find the values of b_1 , b_2 and b_3 that make the system of equations hold. To do this, we can apply Gaussian elimination. This process starts by subtracting a multiple of the first equation from the remaining equations so that b_1 is eliminated from them. We see here that subtracting 2 times the first equation from the second equation should eliminate b_1 . Similarly, subtracting -2 times the first equation from the third equation will again eliminate b_1 . The result after these first two operations is:

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 4 & 4 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -4 \end{bmatrix} \quad (33)$$

The first coefficient in the first equation, the 2, is known as the *pivot* in the first elimination step. We continue the process by eliminating b_2 in all equations after the second, which in this case is just the third equation. This is done by subtracting 4 times the second equation from the third equation to produce:

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & -4 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -4 \end{bmatrix} \quad (34)$$

In this step the 1, the second coefficient in the second equation, was the pivot. We have now created an *upper triangular* matrix, one where all elements below the main diagonal are zero. This system can now be easily solved by *back-substitution*. It is apparent from the last equation that $b_3 = 1$. Substituting this into the second equation yields $b_2 = -2$. Substituting $b_3 = 1$ and $b_2 = -2$ into the first equation we find that $b_1 = 1$.

Gaussian elimination is done in MATLAB with the left division operator `\` (for more information about `\` type `help mldivide` at the command line). Our example problem can be easily solved as follows:

```
>>X = [2 1 1; 4 3 4; -4 2 2]; y = [1; 2; -6];
>>b = X\y
```

b =

```

     1
    -2
     1
```

It is easy to see how this algorithm can be extended to a system of n equation in n unknowns. Multiples of the first equation are subtracted from the remaining equations to produce zero coefficients below the first pivot. Multiples of the second equation are then used to clear the coefficients below the second pivot, and so on. This is repeated until the last equation contains only the last unknown and can be easily solved. This result is substituted into the second to last equation to get the second to last unknown. This is repeated until the entire system is solved.

Singular Matrices and the Concept of Rank

Gaussian elimination appears straightforward enough, however, are there some conditions under which the algorithm may break down? It turns out that, yes, there are. Some of these conditions are easily fixed while others are more serious. The most common problem occurs when one of the pivots is found to be zero. If this is the case, the coefficients below the pivot cannot be eliminated by subtracting a multiple of the pivot equation. Sometimes this is easily fixed: an equation from below the pivot equation with a non-zero coefficient in the pivot column can be swapped with the pivot equation and elimination can proceed. However, if all of the coefficients below the zero pivot are also zero, elimination cannot proceed. In this case, the matrix is called *singular* and there is no hope for a unique solution to the problem.

As an example, consider the following system of equations:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 5 \\ 3 & 6 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 7 \end{bmatrix} \quad (35)$$

Elementary row operations can be used to produce the system:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 3 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 4 \end{bmatrix} \quad (36)$$

This system has no solution as the second equation requires that $b_3 = 1/3$ while the third equation requires that $b_3 = -2$. If we tried this problem in MATLAB we would see the following:

```
»X = [1 2 1; 2 4 5; 3 6 1]; y = [1; 1; 4];
»b = X\y
```

```
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.745057e-18.
```

```
ans =
```

```
1.0e+15 *

2.0016
-1.0008
-0.0000
```

We will discuss the meaning of the warning message shortly.

On the other hand, if the right hand side of the system were changed so that we started with:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 5 \\ 3 & 6 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \\ 7 \end{bmatrix} \quad (37)$$

This would be reduced by elementary row operations to:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 3 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -6 \\ 4 \end{bmatrix} \quad (38)$$

Note that the final two equations have the same solution, $b_3 = -2$. Substitution of this result into the first equation yields $b_1 + 2b_2 = 3$, which has infinitely many solutions. We can see that singular matrices can cause us to have no solution or infinitely many solutions. If we do this problem in MATLAB we obtain:

```
»X = [1 2 1; 2 4 5; 3 6 1]; y = [1; -4; 7];
»b = X\y
```

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.745057e-18.

```
ans =
```

```
-14.3333
 8.6667
-2.0000
```

The solution obtained agrees with the one we calculated above, *i.e.* $b_3 = -2$. From the infinite number of possible solutions for b_1 and b_2 MATLAB chose the values -14.3333 and 8.6667, respectively.

Consider once again the matrix from our previous example and its reduced form:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 5 \\ 3 & 6 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 3 \\ 0 & 0 & -2 \end{bmatrix} \quad (39)$$

We could have taken the elimination one step further by subtracting $-2/3$ of the second equation from the third to produce:

$$\begin{bmatrix} 1 & 3 & 2 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{bmatrix} \quad (40)$$

This is known as the *echelon form* of a matrix. It is necessarily upper triangular, that is, all non-zero entries are confined to be on, or above, the main diagonal (the pivots are the 1 in the first row and the 3 in the second row). The non-zero rows come first, and below each pivot is a column of zeros. Also, each pivot lies to the right of the pivot in the row above. The number of non-zero rows obtained by reducing a matrix to echelon form is the *rank* of the matrix. Note that this procedure can be done on any matrix; it need not be square. It can be shown that the rank of a m by n matrix has to be less than, or equal to, the lesser of m or n , *i.e.*

$$\text{rank}(\mathbf{X}) \leq \min(m,n) \quad (41)$$

A matrix whose rank is equal to the lessor of m or n is said to be of *full rank*. If the rank is less than $\min(m,n)$, the matrix is *rank deficient* or *collinear*. MATLAB can be used to determine the rank of a matrix. In this case we get:

```
»rank(X)
```

```
ans =
```

```
2
```

which is just what we expect.

Matrix Inverses

The *inverse* of matrix \mathbf{A} , \mathbf{A}^{-1} , is the matrix such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$ and $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. It is assumed here that \mathbf{A} , \mathbf{A}^{-1} and \mathbf{I} are all square and of the same dimension. Note, however, that \mathbf{A}^{-1} might not exist. If it does, \mathbf{A} is said to be *invertible*, and its inverse \mathbf{A}^{-1} will be unique.

It can also be shown that the product of invertible matrices has an inverse, which is the product of the inverses in reverse order:

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1} \quad (42)$$

This can be extended to hold with three or more matrices, for example:

$$(\mathbf{ABC})^{-1} = \mathbf{C}^{-1}\mathbf{B}^{-1}\mathbf{A}^{-1} \quad (43)$$

How do we find matrix inverses? We have already seen that we can perform elementary row operations to reduce a matrix \mathbf{A} to an upper triangular echelon form. However, provided that the matrix \mathbf{A} is nonsingular, this process can be continued to produce a diagonal matrix, and finally, the identity matrix. This same set of row operations, if performed on the identity matrix produces the inverse of \mathbf{A} , \mathbf{A}^{-1} . Said another way, the same set of row operations that transform \mathbf{A} to \mathbf{I} transform \mathbf{I} to \mathbf{A}^{-1} . This process is known as the *Gauss-Jordan method*.

Let us consider an example. Suppose that we want to find the inverse of

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 3 & 4 \\ -4 & 2 & 2 \end{bmatrix} \quad (44)$$

We will start by augmenting \mathbf{A} with the identity matrix, then we will perform row operations to reduce \mathbf{A} to echelon form:

$$\left[\begin{array}{ccc|ccc} 2 & 1 & 1 & 1 & 0 & 0 \\ 4 & 3 & 4 & 0 & 1 & 0 \\ -4 & 2 & 2 & 0 & 0 & 1 \end{array} \right] \rightarrow \left[\begin{array}{ccc|ccc} 2 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & -2 & 1 & 0 \\ 0 & 0 & -4 & 10 & -4 & 1 \end{array} \right] \rightarrow$$

We now continue the operation by eliminating the coefficients above each of the pivots. This is done by subtracting the appropriate multiple of the third equation from the first two, then subtracting the second equation from the first:

$$\left[\begin{array}{ccc|ccc} 2 & 1 & 0 & \frac{7}{2} & -1 & \frac{1}{4} \\ 0 & 1 & 0 & 3 & -1 & \frac{1}{2} \\ 0 & 0 & -4 & 10 & -4 & 1 \end{array} \right] \rightarrow \left[\begin{array}{ccc|ccc} 2 & 0 & 0 & \frac{1}{2} & 0 & -\frac{1}{4} \\ 0 & 1 & 0 & 3 & -1 & \frac{1}{2} \\ 0 & 0 & -4 & 10 & -4 & 1 \end{array} \right] \rightarrow$$

Finally, each equation is multiplied by the inverse of the diagonal to get to the identity matrix:

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{1}{4} & 0 & -\frac{1}{8} \\ 0 & 1 & 0 & 3 & -1 & \frac{1}{2} \\ 0 & 0 & 1 & -\frac{5}{2} & 1 & -\frac{1}{4} \end{array} \right] \quad (45)$$

The matrix on the right is \mathbf{A}^{-1} , the inverse of \mathbf{A} . Note that our algorithm would have failed if we had encountered an incurable zero pivot, *i.e.* if \mathbf{A} was singular. In fact, *a matrix is invertible if, and only if, it is nonsingular.*

This process can be repeated in MATLAB using the `rref` (reduced row echelon form) command. We start by creating the \mathbf{A} matrix. We then augment it with the identity matrix using the `eye` command and use the `rref` function to create the reduced row echelon form. In the last step we verify that the 3 by 3 matrix on the right is the inverse of \mathbf{A} . (The result is presented here as fractions rather than decimals by setting the MATLAB command window display preference to “rational”).

```
»A = [2 1 1; 4 3 4; -4 2 2];
»B = rref([A eye(3)])
```

B =

1	0	0	1/4	0	-1/8
0	1	0	3	-1	1/2
0	0	1	-5/2	1	-1/4

```
»A*B(:,4:6)
```

ans =

1	0	0
0	1	0
0	0	1

A simpler way of obtaining an inverse in MATLAB is to use the `inv` command:

```
»Ainv = inv(A)
```

```
Ainv =
```

```
    0.2500    0.0000   -0.1250  
    3.0000   -1.0000    0.5000  
   -2.5000    1.0000   -0.2500
```

Here the results are shown in short, rather than rational format. (Note that, in rational format, a * may appear in place of the 0 in the first row due to roundoff error.) Another useful property of inverses is:

$$(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T \quad (46)$$

We could also verify this for our example in MATLAB:

```
»inv(A') - inv(A)'
```

```
ans =
```

```
1.0e-15 *  
  
   -0.3331    0.4441         0  
    0.1110         0         0  
   -0.0555    0.1110         0
```

which is the zero matrix to within machine precision.

Vector Spaces and Subspaces

Before continuing on it is useful to introduce the concept of a *vector space*. The most important vector spaces are denoted \mathbf{R}^1 , \mathbf{R}^2 , \mathbf{R}^3 ,; there is one for every integer. The space \mathbf{R}^n consists of all column vectors with n (real) components. The exponent n is said to be the *dimension* of the space. For instance, the \mathbf{R}^3 space can be thought of as our “usual” three-dimensional space: each component of the vector is one of the x , y or z axis coordinates. Likewise, \mathbf{R}^2 defines a planar space, such as the x - y plane. A *subspace* is a vector space which is contained within another. For instance, a plane within \mathbf{R}^3 defines a two dimensional subspace of \mathbf{R}^3 , and is a vector space in its own right. In general, a subspace of a vector space is a subset of the space where the sum of any two vectors in the subspace is also in the subspace and any scalar multiple of a vector in the subspace is also in the subspace.

Linear Independence and Basis Sets

In a previous section we defined the rank of a matrix in a purely computational way: the number of non-zero pivots after a matrix had been reduced to echelon form. We are beginning to get the idea that rank and singularity are important concepts when working with systems of equations. To more fully explore these relationships the concept of *linear*

independence must be introduced. Given a set of vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$, if all non-trivial combinations of the vectors are nonzero

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \dots + c_k\mathbf{v}_k \neq \mathbf{0} \quad \text{unless} \quad c_1 = c_2 = \dots = c_k = 0 \quad (47)$$

then the vectors are *linearly independent*. Otherwise, at least one of the vectors is a linear combination of the other vectors and they are *linearly dependent*.

It is easy to visualize linear independence. For instance, any two vectors that are not scalar multiples of each other are linearly independent and define a plane. A third vector in that plane must be linearly dependent as it can be expressed as a weighted sum of the other two vectors. Likewise, any 4 vectors in \mathbf{R}^3 must be linearly dependent. Stated more formally, a set of k vectors in \mathbf{R}^m must be linearly dependent if $k > m$. It is also true that the r nonzero rows of an echelon matrix are linearly independent, and so are the r columns that contain the pivots.

A set of vectors $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k$, in \mathbf{R}^n is said to *span the space* if every vector \mathbf{v} in \mathbf{R}^n can be expressed as a linear combination of \mathbf{w} 's, *i.e.*

$$\mathbf{v} = c_1\mathbf{w}_1 + c_2\mathbf{w}_2 + \dots + c_k\mathbf{w}_k \quad \text{for some } c_i. \quad (48)$$

Note that for the set of \mathbf{w} 's to span \mathbf{R}^n then $k \geq n$. A *basis* for a vector space is a set of vectors which are linearly independent and span the space. Note that the number of vectors in the basis must be equal to the dimension of the space. Any vector in the space can be specified as one and only one combination of the basis vectors. Any linearly independent set of vectors can be extended to a basis by adding (linearly independent) vectors so that the set spans the space. Likewise, any spanning set of vectors can be reduced to a basis by eliminating linearly dependent vectors.

Row Spaces, Column Spaces and Nullspaces

Suppose that \mathbf{U} is the echelon matrix obtained by performing elimination on the m by n matrix \mathbf{A} . Recall that the rank r of \mathbf{A} is equal to the number of nonzero rows of \mathbf{U} . Likewise, the dimension of the *row space* of \mathbf{A} (the space spanned by the rows of \mathbf{A}), denoted $R(\mathbf{A}^T)$, is equal to r , and the rows of \mathbf{U} form a basis for the row space of \mathbf{A} , that is, they span the same space. This is true because elementary row operations leave the row space unchanged.

The *column space* of \mathbf{A} (the space spanned by the columns of \mathbf{A} , also referred to as the range of \mathbf{A}), denoted $R(\mathbf{A})$, also has dimension r . This implies that the number of independent columns in \mathbf{A} equals the number of independent rows, r . A basis can be constructed for the column space of \mathbf{A} by selecting the r columns of \mathbf{U} which contain non-zero pivots.

The fact that the dimension of the row space is equal to the dimension of the column space is one of the most important theorems in linear algebra. This fact is often expressed simply as “row rank = column rank.” This fact is generally not obvious for a random non-square matrix. For square matrices it implies that, if the rows are linearly independent, so are the columns. Note also that the column space of \mathbf{A} is equal to the row space of \mathbf{A}^T .

The *nullspace* of \mathbf{A} , $N(\mathbf{A})$, is of dimension $n - r$. $N(\mathbf{A})$ is the space of \mathbf{R}^n not spanned by the rows of \mathbf{A} . Likewise, the nullspace of \mathbf{A}^T , $N(\mathbf{A}^T)$, (also known as the left nullspace of \mathbf{A}) has dimension $m - r$, and is the space of \mathbf{R}^m not spanned by the columns of \mathbf{A} .

Orthogonality of Subspaces

Earlier we defined orthogonal vectors. Recall that any two vectors \mathbf{v} and \mathbf{w} are orthogonal if their inner product is zero. Two subspaces V and W can also be orthogonal, provided that every vector \mathbf{v} in V is orthogonal to every vector \mathbf{w} in W , *i.e.* $\mathbf{v}^T \mathbf{w} = 0$ for all \mathbf{v} and \mathbf{w} .

Given the definition of orthogonality of subspaces, we can now state more clearly the properties of the row space, column space and nullspaces defined in the previous section. For any m by n matrix \mathbf{A} , the nullspace $N(\mathbf{A})$ and the row space $R(\mathbf{A}^T)$ are orthogonal subspaces of \mathbf{R}^n . Likewise, the left nullspace $N(\mathbf{A}^T)$ and the column space $R(\mathbf{A})$ are orthogonal subspaces of \mathbf{R}^m .

The *orthogonal complement* of a subspace V of \mathbf{R}^n is the space of all vectors orthogonal to V and is denoted V^\perp (pronounced V perp).

Projections onto Lines

The projection of points onto lines is very important in many chemometric methods. In general, the projection problem is, given a vector \mathbf{x} and a point defined by the vector \mathbf{y} , find the point \mathbf{p} along the direction defined by \mathbf{x} which is closest to \mathbf{y} . This is illustrated graphically in Figure 2. Finding this point \mathbf{p} is relatively straightforward once we realize that \mathbf{p} must be a scalar multiple of \mathbf{x} , *i.e.* $\mathbf{p} = \hat{\mathbf{b}}\mathbf{x}$, (here we use the hat symbol $\hat{}$ to denote estimated variables such as $\hat{\mathbf{b}}$), and that the line connecting \mathbf{y} to \mathbf{p} must be perpendicular to \mathbf{x} , *i.e.*

$$\mathbf{x}^T(\mathbf{y} - \hat{\mathbf{b}}\mathbf{x}) = 0 \rightarrow \mathbf{x}^T\mathbf{y} = \hat{\mathbf{b}}\mathbf{x}^T\mathbf{x} \rightarrow \hat{\mathbf{b}} = \frac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x}} \quad (49)$$

Therefore, the projection \mathbf{p} of the point \mathbf{y} onto the line spanned by the vector \mathbf{x} is given by

$$\mathbf{p} = \hat{\mathbf{b}}\mathbf{x} = \frac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x}} \mathbf{x} \quad (50)$$

The (squared) distance from \mathbf{y} to \mathbf{p} can also be calculated. The result is

$$\|\mathbf{y} - \mathbf{p}\|^2 = \frac{(\mathbf{y}^T\mathbf{y})(\mathbf{x}^T\mathbf{x}) - (\mathbf{x}^T\mathbf{y})^2}{\mathbf{x}^T\mathbf{x}} \quad (51)$$

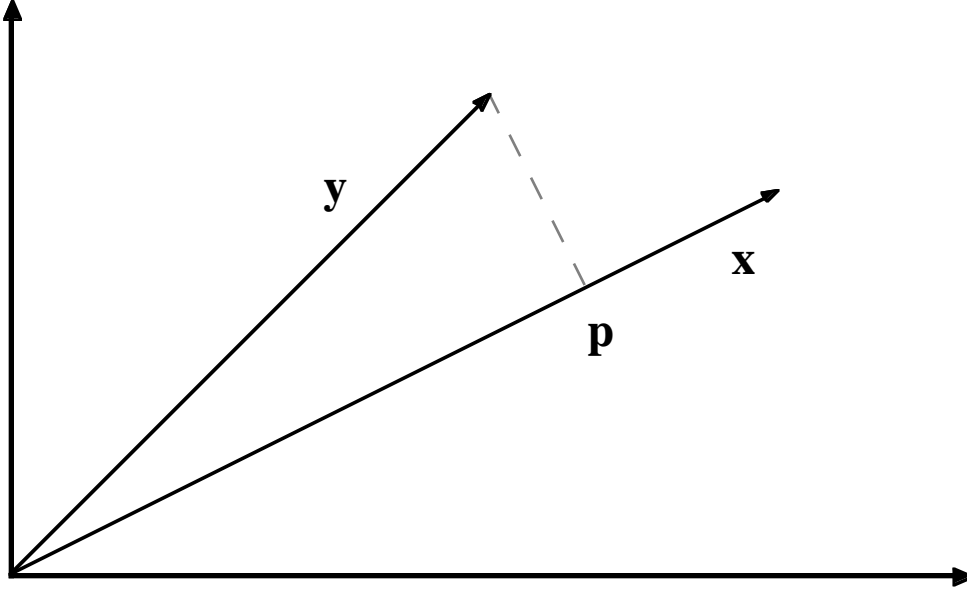


Figure 2. The Projection of the vector \mathbf{y} onto the vector \mathbf{x} .

We may also be interested in the projection of a point \mathbf{y} onto a subspace. For instance, we may want to project the point onto the plane defined by two vectors, or an n dimensional subspace defined by a collection of vectors, *i.e.* a matrix. In this case, the vector \mathbf{x} in the two equations above will become the matrix \mathbf{X} , and the inner product $\mathbf{x}^T \mathbf{x}$ will become $\mathbf{X}^T \mathbf{X}$. If \mathbf{X} is m by n and rank r , the matrix $\mathbf{X}^T \mathbf{X}$ is n by n , symmetric ($\mathbf{X}^T \mathbf{X} = (\mathbf{X}^T \mathbf{X})^T$) and also of rank r . Thus we see that if \mathbf{X} has linearly independent columns, *i.e.* rank $r = n$, then $\mathbf{X}^T \mathbf{X}$ is nonsingular and therefore invertible. This is a very important result as we shall see shortly.

Projections onto Subspaces and Least Squares

Previously we have considered solving $\mathbf{X}\mathbf{b} = \mathbf{y}$ only for systems where there are the same number of equations as variables ($m = n$). This system either has a solution or it doesn't. We now consider the case of $m > n$, where we have more samples (rows) than variables (columns), with the possibility that the system of equations can be inconsistent, *i.e.* there is no solution which makes the equality hold. In these cases, one can only solve the system such that the average error in the equations is minimized.

Consider for a moment the single variable case $\mathbf{x}\mathbf{b} = \mathbf{y}$ where there is more than 1 equation. The error is nothing more than the length of the vector $\mathbf{x}\mathbf{b} - \mathbf{y}$, $E = \|\mathbf{x}\mathbf{b} - \mathbf{y}\|$. It is easier, however, to minimize the squared length,

$$E^2 = (\mathbf{x}\mathbf{b} - \mathbf{y})^T (\mathbf{x}\mathbf{b} - \mathbf{y}) = \mathbf{x}^T \mathbf{x} \mathbf{b}^2 - 2\mathbf{x}^T \mathbf{y} \mathbf{b} + \mathbf{y}^T \mathbf{y} \quad (52)$$

The minimum of this can be found by taking the derivative with respect to \mathbf{b} and setting it to zero:

$$\frac{dE^2}{d\mathbf{b}} = 2\mathbf{x}^T \mathbf{x} \mathbf{b} - 2\mathbf{x}^T \mathbf{y} = 0 \quad (53)$$

The least squares solution to the problem is then

$$\hat{\mathbf{b}} = \frac{\mathbf{x}^T \mathbf{y}}{\mathbf{x}^T \mathbf{x}} \quad (54)$$

This is the same solution we obtained to the projection problem $\mathbf{p} = \hat{\mathbf{b}}\mathbf{x}$ in equation (49). Thus, requiring that the error vector be perpendicular to \mathbf{x} gave the same result as the least squares approach. This suggests that we could solve systems in more than one variable by either geometry or calculus.

Consider now the system $\mathbf{X}\mathbf{b} = \mathbf{y}$, where \mathbf{X} is m by n , $m > n$. Let us solve this system by requiring that $\mathbf{X}\hat{\mathbf{b}} - \mathbf{y}$ be perpendicular to the column space of \mathbf{X} . For this to be true, each vector in the column space of \mathbf{X} must be perpendicular to $\mathbf{X}\hat{\mathbf{b}} - \mathbf{y}$. Each vector in the column space of \mathbf{X} , however, is expressible as a linear combination, say \mathbf{c} , of the columns of \mathbf{X} , $\mathbf{X}\mathbf{c}$. Thus, for all choices of \mathbf{c} ,

$$(\mathbf{X}\mathbf{c})^T(\mathbf{X}\hat{\mathbf{b}} - \mathbf{y}) = 0, \quad \text{or} \quad \mathbf{c}^T[\mathbf{X}^T\mathbf{X}\hat{\mathbf{b}} - \mathbf{X}^T\mathbf{y}] = 0 \quad (55)$$

There is only one way that this can be true for all \mathbf{c} : the vector in the brackets must be zero. Therefore, it must be true that

$$\mathbf{X}^T\mathbf{X}\hat{\mathbf{b}} = \mathbf{X}^T\mathbf{y} \quad (56)$$

These are known as the *normal equations* for least squares problems. We can now solve for $\hat{\mathbf{b}}$ by multiplying through by $(\mathbf{X}^T\mathbf{X})^{-1}$ to obtain:

$$\hat{\mathbf{b}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad (57)$$

The projection of \mathbf{y} onto the column space of \mathbf{X} is therefore

$$\mathbf{p} = \mathbf{X}\hat{\mathbf{b}} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad (58)$$

In chemometrics we commonly refer to $\hat{\mathbf{b}}$ as the regression vector. Often \mathbf{X} consists of the matrix of m samples with n measured variables. The vector \mathbf{y} consists of some property (such as a concentration or quality parameter) for the m samples. We then want to determine a $\hat{\mathbf{b}}$ that we can use with new samples, \mathbf{X}_{new} , to estimate their properties $\hat{\mathbf{y}}_{\text{new}}$:

$$\hat{\mathbf{y}}_{\text{new}} = \mathbf{X}_{\text{new}}\hat{\mathbf{b}} \quad (59)$$

As an example, suppose we measured two variables, x_1 and x_2 on four samples, and also had a quality variable y . Our data would then be:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 2 & 1 \\ 2 & 2 \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 6 \\ 6 \\ 7 \\ 11 \end{bmatrix} \quad (60)$$

We can determine a regression vector $\hat{\mathbf{b}}$ that relates the measured variables x_1 and x_2 to the quality variable y in MATLAB as follows:

```

»X = [1 1; 1 2; 2 1; 2 2]; y = [6 6 7 11]';
»b = inv(X'*X)*X'*y

```

b =

```

    3.0000
    2.0000

```

Thus, our model for estimating our quality variable is $y = 3x_1 + 2x_2$. In MATLAB, however, we can simplify this calculation by using the “left division” symbol `\` instead of the `inv` operator and get the same result:

```

»b = X\y

```

b =

```

    3.0000
    2.0000

```

The $\hat{\mathbf{b}}$ can now be used to get \mathbf{p} , the projection of \mathbf{y} into \mathbf{X} , and the difference between \mathbf{y} and its projection into \mathbf{X} , \mathbf{d} :

```

»p = X*b

```

p =

```

     5
     7
     8
    10

```

```

»d = y-p

```

d =

```

     1
    -1
    -1
     1

```

We can also check to see if \mathbf{d} is orthogonal to the columns of \mathbf{X} . The simplest way to do this is to check the value of $\mathbf{X}^T\mathbf{d}$:

```
»X'*d
```

```
ans =
```

```
1.0e-14 *
```

```
-0.9770
```

```
-0.9770
```

This gives us the inner product of each of the columns of \mathbf{X} with \mathbf{d} . We see that the values are very close to zero, the only difference from zero being due to machine round off error. Thus, \mathbf{d} is orthogonal to the columns of \mathbf{X} , as it should be.

III-conditioned Problems

We have now seen how to solve least squares problems in several variables. Once again, we might ask: “when might I expect this procedure to fail?” Based on our previous experience with the solution of m equations in m unknowns and the formation of inverses, we expect that the least squares procedure will fail when \mathbf{X} is rank deficient, that is, when $\text{rank}(\mathbf{X}^T\mathbf{X}) < n$. However, what about when \mathbf{X} is nearly rank deficient, *i.e.* when one column of \mathbf{X} is nearly a linear combination of the other columns? Let us consider such a system:

$$\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \\ 4 & 8.0001 \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix} \quad (61)$$

Here the second column of \mathbf{X} is almost, but not quite, twice the first column. In other words, \mathbf{X} is close to being singular. We can now use MATLAB to estimate a regression vector $\hat{\mathbf{b}}$ for this problem:

```
»X = [1 2; 2 4; 3 6; 4 8.0001]; y = [2 4 6 8]';
```

```
»b = X\y
```

```
b =
```

```
2
```

```
0
```

Here we see that the system can be solved exactly by $y = 2x_1$. However, what would happen if the y data changed a very small amount? Consider changing the third element of y from 6 to 5.9999 and 6.0001. The following results:

»X = [1 2; 2 4; 3 6; 4 8.0001]; y = [2 4 6.0001 8]'; b = X\y

b =

3.7143
-0.8571

»X = [1 2; 2 4; 3 6; 4 8.0001]; y = [2 4 5.9999 8]'; b = X\y

b =

0.2857
0.8571

We see that the regression coefficients changed from $\hat{\mathbf{b}} = [3.7143 \ -0.8571]$ to $\hat{\mathbf{b}} = [0.2857 \ 0.8571]$ given this very small change in the data. This is an example of what happens in regression when the \mathbf{X} matrix is very close to singular. Methods for dealing with these ill-conditioned problems will be discussed in the next section.

Projection Matrices

We saw previously that given the problem $\mathbf{X}\mathbf{b} = \mathbf{y}$, we could obtain the projection of \mathbf{y} onto the columns of \mathbf{X} , \mathbf{p} , from:

$$\mathbf{p} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad (62)$$

$$\mathbf{p} = \mathbf{P}\mathbf{y} \quad (63)$$

The quantity $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$, which we shall denote \mathbf{P} , is a *projection matrix*. Projection matrices have two important properties: they are idempotent and symmetric. Thus

$$\mathbf{P}\mathbf{P} = \mathbf{P}^2 = \mathbf{P} \quad (64)$$

$$\mathbf{P}^T = \mathbf{P} \quad (65)$$

It is easy to see that \mathbf{P} must be idempotent: the first application of \mathbf{P} projects \mathbf{y} into \mathbf{X} . A second application won't change the result since $\mathbf{P}\mathbf{y}$ will already be in the column space of \mathbf{X} .

Orthogonal and Orthonormal Bases

When we think of a basis for a space, such as the x-y plane, we normally think of an orthogonal basis, *i.e.* we define our distances in the plane based on two directions that are perpendicular. Orthogonal basis sets, it turns out, are convenient from both a conceptual and mathematical perspective. Basis sets can be made even simpler by normalizing the basis vectors to unit length. The result is an orthonormal basis. An orthonormal basis, $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ has the property that:

$$\mathbf{v}_i^T \mathbf{v}_j = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases} \quad (66)$$

Of course, we are most familiar with the standard basis, $\mathbf{v}_1 = [1 \ 0 \ \dots \ 0]^T$, $\mathbf{v}_2 = [0 \ 1 \ \dots \ 0]^T$, etc., however, it is possible to construct an infinite number of orthonormal bases for any space of dimension 2 or greater by simply rotating the basis vectors while keeping them at right angles to each other.

Suppose for a moment that we are interested in projecting a vector \mathbf{y} onto an \mathbf{X} that consists of orthonormal columns. In this case, $\mathbf{X}^T \mathbf{X} = \mathbf{I}$, thus

$$\mathbf{P} = \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T = \mathbf{X} \mathbf{X}^T \quad (67)$$

A square matrix with orthonormal columns is known as an *orthogonal matrix* (though orthonormal matrix would have been a better name). An orthogonal matrix \mathbf{Q} has the following properties:

$$\mathbf{Q}^T \mathbf{Q} = \mathbf{I}, \quad \mathbf{Q} \mathbf{Q}^T = \mathbf{I}, \quad \text{and} \quad \mathbf{Q}^T = \mathbf{Q}^{-1} \quad (68)$$

Furthermore, \mathbf{Q} will also have orthonormal rows, *i.e.* if \mathbf{Q} is an orthogonal matrix, so is \mathbf{Q}^T . We will see this again shortly when we consider Principal Components Analysis (PCA) in the next section.

Pseudoinverses

We saw previously that we were not able to solve $\mathbf{X}\mathbf{b} = \mathbf{y}$ using the “normal equations” in the case where the columns of \mathbf{X} were not independent or collinear. But is that the best we can do? Are we simply forced to conclude that there is no solution to the problem? In fact, it is still possible to obtain a least squares solution to the problem. To do so, we must introduce the concept of the pseudoinverse, \mathbf{X}^+ , for a matrix \mathbf{X} that is not invertible. The general situation when \mathbf{X} is collinear is that there are many solutions to $\mathbf{X}\mathbf{b} = \mathbf{y}$. Which one should we choose?: The one with minimum length of $\hat{\mathbf{b}}$, $\|\hat{\mathbf{b}}\|$. To get this we require that the component of $\hat{\mathbf{b}}$ in the nullspace of \mathbf{X} be zero, which is the same as saying that $\hat{\mathbf{b}}$ must lie in the row space of \mathbf{X} . Thus, the optimal least squares solution to any $\mathbf{X}\mathbf{b} = \mathbf{y}$ is $\hat{\mathbf{b}}$ such that:

$\mathbf{X}\hat{\mathbf{b}}$ equals the projection of \mathbf{y} into the column space of \mathbf{X} , and

$\hat{\mathbf{b}}$ lies in the row space of \mathbf{X} .

We call the matrix that solves this problem the pseudoinverse \mathbf{X}^+ , where $\hat{\mathbf{b}} = \mathbf{X}^+ \mathbf{y}$. Note that when the columns of \mathbf{X} are independent,

$$\mathbf{X}^+ = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \quad (69)$$

This satisfies our requirements stated above. In the case where \mathbf{X} is collinear, we must find another way to calculate \mathbf{X}^+ .

The Singular Value Decomposition

We now introduce the *singular value decomposition*, or SVD, which will allow us to obtain a general solution to the problem of finding pseudoinverses. Any m by n matrix \mathbf{X} can be factored into

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T \quad (70)$$

where \mathbf{U} is orthogonal and m by m , \mathbf{V} is orthogonal and n by n , and \mathbf{S} is m by n and diagonal. The non-zero entries of \mathbf{S} are referred to as the *singular values* and decrease monotonically from the upper left to the lower right of \mathbf{S} .

As an example of the SVD, let us consider the matrix

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 5 \\ 3 & 5 & 8 \\ 4 & 8 & 12 \end{bmatrix} \quad (71)$$

Here we have chosen \mathbf{X} to be collinear: the third column is the sum of the first two columns. We can now use MATLAB to compute the SVD of \mathbf{X} , and verify that the product of the \mathbf{U} , \mathbf{S} and \mathbf{V}^T reproduces \mathbf{X} .

```
»X = [1 2 3; 2 3 5; 3 5 8; 4 8 12];  
»[U,S,V] = svd(X)
```

U =

0.1935	0.1403	-0.9670	0.0885
0.3184	-0.6426	0.0341	0.6961
0.5119	-0.5022	-0.0341	-0.6961
0.7740	0.5614	0.2503	0.1519

S =

19.3318	0	0
0	0.5301	0
0	0	0.0000
0	0	0

V =

0.2825	-0.7661	0.5774
0.5221	0.6277	0.5774
0.8047	-0.1383	-0.5774


```
»U*S*V'
```

```
ans =
```

```
1.0000    2.0000    3.0000
2.0000    3.0000    5.0000
3.0000    5.0000    8.0000
4.0000    8.0000   12.0000
```

Note that \mathbf{S} is diagonal, and that the diagonal elements decrease from upper left to lower right. In fact, the last singular value appears to be zero. Recall now that the inverse of a product is the product of the inverses in reverse order. This leads to an explicit formula for the pseudoinverse of \mathbf{X} , \mathbf{X}^+ :

$$\mathbf{X}^+ = \mathbf{V}\mathbf{S}^+\mathbf{U}^T \quad (72)$$

Here we must be very careful in formation of \mathbf{S}^+ . In practice, all singular values that are close to zero are set to zero in the inverse (the challenge comes with determining what constitutes “close to zero”). Alternately, one can think of this as truncation of the matrices \mathbf{U} , \mathbf{S} and \mathbf{V} where only the first r columns of \mathbf{U} and \mathbf{V} are retained, along with the first r rows and columns of \mathbf{S} , where $r = \text{rank}(\mathbf{X})$. In MATLAB, we see for our example:

```
»U(:,1:2)*S(1:2,1:2)*V(:,1:2)'
```

```
ans =
```

```
1.0000    2.0000    3.0000
2.0000    3.0000    5.0000
3.0000    5.0000    8.0000
4.0000    8.0000   12.0000
```

It is apparent that we only need the first two “factors” in the SVD to reproduce the original data. The inverse of \mathbf{X} , \mathbf{X}^+ can then be calculated from:

```
»Xinv = V(:,1:2)*inv(S(1:2,1:2))*U(:,1:2)'
```

```
Xinv =
```

```
-0.2000    0.9333    0.7333   -0.8000
 0.1714   -0.7524   -0.5810    0.6857
-0.0286    0.1810    0.1524   -0.1143
```

Note that this is same result as would be obtained from the MATLAB command `pinv`, which uses the SVD and sets all singular values that are within machine precision to zero:

```
»pinv(X)
```

```
ans =
```

```
-0.2000    0.9333    0.7333   -0.8000  
 0.1714   -0.7524   -0.5810    0.6857  
-0.0286    0.1810    0.1524   -0.1143
```

Let us now return to our ill-conditioned regression problem where

$$\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \\ 4 & 8.0001 \end{bmatrix} \text{ and } \mathbf{y} = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix} \quad (73)$$

Recall that a small change in the third element of \mathbf{y} produced wildly different regression vectors. We can now repeat this problem assuming that the true rank of \mathbf{X} is 1, *i.e.* to within measurement accuracy the second column of \mathbf{X} is a multiple of the first column. In MATLAB we would obtain:

```
»X = [1 2; 2 4; 3 6; 4 8.0001]; y = [2 4 6 8]';  
»[U,S,V] = svd(X);  
»Xinv = V(:,1)*inv(S(1,1))*U(:,1)'
```

```
Xinv =
```

```
0.0067    0.0133    0.0200    0.0267  
0.0133    0.0267    0.0400    0.0533
```

```
»b = Xinv*y
```

```
b =
```

```
0.4000  
0.8000
```

With this inverse, changing the third element of \mathbf{y} would no longer change the result significantly:

```
»y = [2 4 5.9999 8]';  
»b = Xinv*y
```

b =

```
0.4000  
0.8000
```

```
»y = [2 4 6.0001 8]';  
»b = Xinv*y
```

b =

```
0.4000  
0.8000
```

Thus we see that the use of a pseudoinverse can enhance the stability of our calculations. Pseudoinverses play an important role in many chemometric regression methods such as Principal Components Regression (PCR) and Partial Least Squares (PLS), and many different approaches to the defining the pseudoinverse are used.

Conclusion

This introduction to linear algebra, while brief, should include most of what the reader needs to know in order to tackle the chemometrics literature. Interested readers are encouraged to consult texts on the subject for more detail on the methods, including proofs and additional examples.

Acknowledgement

This article was adapted from a section of the *PLS_Toolbox 2.0 for use with MATLAB* (Wise and Gallagher, Eigenvector Research, Inc, 1998) software package.