THE UNIVERSITY OF NOTTINGHAM

FACULTY OF APPLIED SCIENCE

Applications of Computers

LECTURE 13

"LINEAR ALGEBRA"

by

M. WOODGER, B.Sc.

LINEAR ALGEBRA

SYNOPSIS

General remarks. Programme checks.

Simultaneous Linear Equations. Gaussian Elimination.

Inversion of Matrices.

Latent roots and vectors of matrices.

General manipulation of matrices: compound problems: need for an interpretive scheme.

1 - INTRODUCTION

1.1 In numerical work "linear algebra" means any calculations restricted to operations upon arrays of numbers like

$$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{matrix}$$

in which multiples of one row or column are added to others. Such arrays are called matrices, and a single row or column is often referred to as a vector. We may use symbols A, x for matrices and vectors respectively. Linear algebra arises naturally in a very wide field of applications, nearly always through consideration of sets of simultaneous linear algebraic equations, that is to say equations such as

$$\begin{aligned} a_{11}u + a_{12}v + a_{13}w &= b_1 \\ a_{21}u + a_{22}v + a_{23}w &= b_2 \\ a_{31}u + a_{32}v + a_{33}w &= b_3 \end{aligned} \qquad (1)$$

where one knows the coefficients a_{ij} and "right-hand sides" b_i and requires the solution u, v, w. The term "linear" is borrowed from geometry, in which such equations describe lines, planes, etc.

1. 2 The problem of inversion is to determine a matrix

$$\begin{matrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{matrix}$$

such that the solution of the equations (1) can be written explicitly as

$$\begin{aligned} u &= c_{11}b_1 + c_{12}b_2 + c_{13}b_3 \\ v &= c_{21}b_1 + c_{22}b_2 + c_{23}b_3 \\ w &= c_{31}b_1 + c_{32}b_2 + c_{33}b_3 \end{aligned} \qquad (2)$$

This is useful when a large number of sets of equations such as (1) have to be solved, in which only the b_i are changed between one set and the next.

The symbol A^{-1} is used for the inverse matrix of A, which has to be square.

1. 3 In the analysis of oscillatory systems, whether electrical or mechanical, one is led to the more difficult "latent root" or "eigenvalue" problem.

Here one has to determine a quantity λ such that

$$\begin{aligned} a_{11}u + a_{12}v + a_{13}w &= \lambda u \\ a_{21}u + a_{22}v + a_{23}w &= \lambda v \\ a_{31}u + a_{32}v + a_{33}w &= \lambda w, \end{aligned} \qquad (3)$$

and for n equations there will in general be just n values of λ, real or complex, for which there is a corresponding solution u, v, w, called the "latent vector" or "eigenvector". λ will be related to a natural frequency of oscillation of the system and u, v, w will represent the corresponding mode of oscillation. The matrix A of coefficients has to be square.

1.4 What makes linear algebra so ideally suited to automatic calculation and so tedious by hand is the uniformity of the arithmetical manipulations required - the same simple sequence of additions and multiplications is constantly repeated.

One can store the elements of a row or a column of a matrix in consecutive storage locations and have a relatively small and fast loop of instructions for dealing with each element as it is selected from store.

1.5 Another important advantage is the very satisfactory programmed arithmetic check known as the "distributive check" which is available. This is achieved by storing the sum $s_1=a_{11}+a_{12}+a_{13}+b_1$ together with the numbers $a_{11}, a_{12}, a_{13}, b_1$ and treating it in the same way. Then, in the simplest case, after forming the multiples ka_{11}, ka_{12}, ka_{13}, kb_1, ks_1 in the course of a calculation, the machine checks that the last one is still equal to the sum of the preceding numbers.

1.6 Taking all digital computers in the country together, it is probably true to say that for about half of their operating time on real problems they are doing linear algebra. Almost the first general purpose programme to be prepared for a new computer is one for solving sets of simultaneous linear algebraic equations, and there is very soon a demand for matrix inversion and latent root programmes as well.

As instances of the origins of such sets of equations may be mentioned the following: (i) Analysis of elastic structures, such as steel frames of buildings or aircraft wings and fuselages. Given the elastic constants (the a_{ij}) and the loading (the b_i) it is required to calculate the consequent deflections (u, v, w).
(ii) Steady flow of heat, fluids, electrical fields, and a wide range of other physical problems are expressible as the solution of partial differential equations of elliptic type.

As far as the computer is concerned these amount to sets of large numbers of linear algebraic equations, usually of rather simple form. The development of very large storage capacity for future computers is stimulated by the need to solve these very large sets of equations, especially in connection with military applications, shock waves, nuclear explosions and the like.

The above two categories have the common property that they arise from the idealisation of a continuous physical system to a discrete mathematical model. Thus in studying the deflection (and vibration) of a beam one replaces it by the simplified "model" of a number of point masses, elastically interconnected, each representing a finite strip of the beam. The larger the number of representative points the more accurate the idealisation, although engineering accuracy is obtained from surprisingly few points.

Likewise in the heat conduction problem one works with the values of the temperature at the corners of the squares of a regular network covering the region of interest, and there is one equation for each point of the net. Again, the finer the net the more accurate are the results, although adequate estimates of thermal gradients, for example in assessing structural safety factors, may be obtained with a fairly coarse net.

(iii) A further common origin of sets of simultaneous linear algebraic equations is statistics. Any regression analysis, and in particular curve-fitting of data at unequally spaced intervals by the method of least squares, leads to such equations.

2 - METHODS

2.1 Gaussian Elimination.

This is the method one learns at school. Starting from equations (1) we eliminate u from the second and third equations by adding $-a_{21}/a_{11}$ times the first row of the matrix to its second row, and also $-a_{31}/a_{11}$ times the first row to the last row. This gives a matrix of the form

$$\begin{matrix} a_{11} & a_{12} & a_{13} & b_1 \\ 0 & a'_{22} & a'_{23} & b'_2 \\ 0 & a'_{32} & a'_{33} & b'_3 \end{matrix}$$

We now eliminate the second variable v from the third equation by adding $-a'_{32}/a'_{22}$ times the second row to the third row and end up with the matrix "reduced to triangular form"

$$\begin{matrix} a_{11} & a_{12} & a_{13} & b_1 \\ 0 & a'_{22} & a'_{23} & b'_2 \\ 0 & 0 & a''_{33} & b''_3 \end{matrix}$$

Finally we determine $w = b''_3/a''_{33}$ from the last equation, use this to substitute in the second equation to get

$$v = (b'_2 - a'_{23}w)/a'_{22},$$

and use v and w to substitute in the first equation to get

$$u = (b_1 - a_{12}v - a_{13}w)/a_{11}.$$

This is the second half of the process and is called "back-substitution".

It is obvious that the method would break down if the so-called "pivots" a_{11} or a'_{22} or a''_{33} were zero, and would cause trouble if any of these were small. A simple precaution, which renders Gaussian Elimination the reliable and universally used numerical procedure that it has become, is to rearrange the equations at each step in the reduction so that the pivot is always the <u>largest number in the first column</u> of the partially reduced matrix. Thus we arrange to begin with that a_{21} and a_{31} are not larger (in absolute magnitude) than a_{11}, and at the next step that a'_{32} is not larger than a'_{22} - otherwise we interchange the last two equations. If small pivots still appear the solution is poorly determinable and the equations are said to be "<u>ill-conditioned</u>".

Other methods, suitable especially for equations of particular forms, will be dealt with if time permits.

<u>2. 2</u> Inversion of matrices.

This is simply an extension of 2. 1 in which one deals simultaneously with right-hand sides

$$\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0, & 0, & 1 \end{array}$$

The three solutions so obtained are the three columns of the inverse matrix. During the reduction one keeps all right-hand sides throughout the working, except where storage space can be saved by not storing zeros. The back-substitution is carried out separately for each column of the inverse.

2.3 Latent roots and vectors of matrices.

The variety of methods available here is due chiefly to the very varied requirements for specific purposes.

In vibration problems one has essentially a symmetric matrix ($a_{21} = a_{12}$, $a_{31} = a_{13}$, $a_{32} = a_{23}$) and the roots must all be real; one wants only the largest few roots and corresponding vectors and the roots are not often very close together. In this case the standard method is to start with any set of numbers $\begin{matrix} u_1 \\ v_1 \\ w_1 \end{matrix}$ and "multiply" by the matrix to get a second set

$$\begin{aligned} u_2 &= a_{11}u_1 + a_{12}v_1 + a_{13}w_1 \\ v_2 &= a_{21}u_1 + a_{22}v_1 + a_{23}w_1 \\ w_2 &= a_{31}u_1 + a_{32}v_1 + a_{33}w_1 \end{aligned}$$

"normalise" this set by dividing through by its largest element (to make the largest one 1), and again multiply by the matrix. This process is repeated until the numbers converge - the result is the latent vector x corresponding to the largest root λ, and the root itself appears as the largest vector element after the matrix multiplication.

One has then to "remove" this root, i.e. to calculate a new matrix B whose roots other than λ are the same as those of the original matrix A, and either (a): B is smaller than A by one row and column, or (b): B is the same size as A and has a zero root in place of λ.

Method (a) is perhaps commoner; it reduces the volume of computation a little and is applicable directly to unsymmetric matrices A. If the normalised latent vector of A is $\begin{matrix}1\\ v\\ w\end{matrix}$ then one forms B by subtracting from the second and third rows of A the multiples v and w respectively of the first row, and then omitting the first row and column. The latent vectors of B have now only two elements and need to be combined with the vector x by a back substitution process to obtain the corresponding latent vectors of A.

Method (b) for symmetric matrices avoids a back substitution step since the latent roots and vectors of B are already appropriate to A. B is formed from A by subtracting λ times 1, v, w from the first row of A, λ v times this from the second row of A, and λ w times this from the last row.

Among other methods of value according to the requirements of the case are those of Jacobi, Lanczos and Givens which will be described if time permits.

3 - MATRIX ALGEBRA IN GENERAL

Occasionally one has already available for the computer the coefficients a_{ij} and b_i, but it is more usual to find that a variety of simple preliminary calculations are necessary in order to obtain them. These calculations generally fall under the heading of "linear algebra", and can be expressed in terms of matrix addition or multiplication, or simplified forms of these operations.

3.1 If the matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$

is added to another one B, which must be of the same shape and size, i.e. of three rows and four columns, the result is got by adding together corresponding elements in the two matrices, thus:

$$A + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} & a_{14} + b_{14} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} & a_{24} + b_{24} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} & a_{34} + b_{34} \end{bmatrix}$$

If A is multiplied in front by a row vector $\begin{bmatrix} p & q & r \end{bmatrix}$, which must have the same number of columns as A has rows, the result is another row vector which is got by adding together p times the first row of A, q times the second row, and r times the third, thus:

$$\begin{bmatrix} p & q & r \end{bmatrix} A = \big[pa_{11} + qa_{21} + ra_{31},\ pa_{12} + qa_{22} + ra_{32},$$
$$pa_{13} + qa_{23} + ra_{33},\ pa_{14} + qa_{24} + ra_{34} \big]$$

We call this row a linear combination of the rows of A with coefficients p, q, r.

If however A is multiplied behind by a column vector $\begin{bmatrix} d \\ e \\ f \\ g \end{bmatrix}$, this must have the same number of rows as A has columns, and the result is another column vector which is got by adding the multiples of the four columns of A given by d, e, f, g respectively, thus:

$$A \begin{bmatrix} d \\ e \\ f \\ g \end{bmatrix} = \begin{bmatrix} a_{11}d + a_{12}e + a_{13}f + a_{14}g \\ a_{21}d + a_{22}e + a_{23}f + a_{24}g \\ a_{31}d + a_{32}e + a_{33}f + a_{34}g \end{bmatrix}$$

This column is a linear combination of the columns of A.

The general case of pre-multiplication of A by a matrix B is just the same, but instead of a single row p q r producing a single row product we have each row of B treated in the same way to produce the same number of rows in the product matrix. This can be expressed by saying that BA is a matrix of as many rows as B, each row being a linear combination of the rows of A, the coefficients in the combination being the elements of the corresponding row of B. For this to be possible B must have as many columns as A has rows.

Likewise post-multiplication of A by B can be regarded as the formation of a matrix by columns, each of which is obtained as a linear combination of the columns of A using coefficients from a column of B. B must have as many rows as A has columns (which is the same condition as before).

When A is pre-multiplied by the square matrix

$$\begin{bmatrix} p & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & r \end{bmatrix}$$

(called a "diagonal" matrix) the result is

$$\begin{bmatrix} pa_{11} & pa_{12} & pa_{13} & pa_{14} \\ qa_{21} & qa_{22} & qa_{23} & qa_{24} \\ ra_{31} & ra_{32} & ra_{33} & ra_{34} \end{bmatrix}$$

i.e. the rows of A are multiplied by the diagonal elements.

Likewise post-multiplication of A by the diagonal matrix

$$\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 0 & g \end{bmatrix}$$

produces

$$\begin{bmatrix} a_{11}d & a_{12}e & a_{13}f & a_{14}g \\ a_{21}d & a_{22}e & a_{23}f & a_{24}g \\ a_{31}d & a_{32}e & a_{33}f & a_{34}g \end{bmatrix}$$

the <u>columns</u> of A being multiplied by the diagonal elements.

In the special case p=q=r we have all elements of A multiplied by p and call this "scalar" multiplication of A by p.

Finally the matrix may be written "transposed", i.e. with columns and rows interchanged thus:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}$$

This is often denoted by A'.

<u>3.2</u> With the aid of the above ideas much of the systematic preliminary processing of numerical data can be effected by linear algebra.

Scaling the columns of a matrix, which corresponds to scaling the variables u, v, w separately in the case of the linear equations (1), is simply post-multiplication by a diagonal matrix composed of the scaling factors. Scaling the rows, which does not affect the result in (1), may be desirable if some rows of coefficients are much smaller than the others, and can be achieved by diagonal matrix pre-multiplication.

Change of origins of u, v, w to p, q, r respectively amounts to subtracting $A \begin{bmatrix} p \\ q \\ r \end{bmatrix}$ from the right-hand side $\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$.

3.3 As a result of the relatively few kinds of operations involved in the manipulation of matrices the subject lends itself particularly well to the use of interpretive programmes. One can do most of the necessary calculations if one has the following repertoire of functions available -

Input a matrix to location a

Output a matrix from location a

Add the matrix at location a to the matrix at location b, and store the result at location c.

Multiply the matrix at location a by the matrix at location b, and store the result at location c.

Transpose the matrix at location a and store the result at location c.

Pre-multiply the diagonal matrix at location a by the matrix at location b and store the result at location c.

Post-multiply the matrix at location a by the diagonal matrix at location b and store the result at location c.

Extract part of the matrix at location a and store the result at location c.

Compound the matrix at location a with the matrix at location b and store the result at location c.

If the dimensions (numbers of rows and columns) of the matrices are stored with them, each sub-programme called into use by the master programme when interpreting a codeword needs only to be given the addresses of the matrices to be operated on and the address to which to send the resultant matrix. Thus a three address code is convenient, and we can arrange for the addresses to refer to track numbers on an auxiliary magnetic drum store, as with DEUCE. A codeword (a, b, c, r) can mean "carry out operation number r using matrix addresses a, b, c".

The distributive checks can be incorporated in the component programmes, but are only possible with precision where fixed point working is used. However, the device of "block-floating" arithmetic can be used, in which a common exponent is associated with each matrix, the exponents of two matrices being taken into account in deciding the exponent of their sum or product. This preserves the automatic scaling advantages of ordinary floating-point arithmetic. Where multiplication is involved the products are accumulated to double-length (two word) accuracy before shifting and rounding off to single length. This means that each component programme (when multiplication is used) has to carry out the operation twice - once to find how big the largest element of the resultant matrix is and hence the shift after multiplication (and the exponent to be associated with the result), and then again to form and store the single-length matrix.

Because each component operation takes some time, the time of interpretation of each codeword is not so important as with other interpretive schemes. It is at present of the order of one second for DEUCE.