Vectorisation in Matrix Algebra [Lesson 5 of Matrix Algebra (Vectorisation)]

Tony Davies

Norwich Near Infrared Consultancy , 75 Intwood Road, Cringleford, Norwich NR4 6AA, UK

Tom Fearn

Department of Statistical Science, University College London, Gower Street, London, WC1E 6BT, UK

Question: Why learn matrix algebra when you can buy programs "off-the-shelf"?
Answer: You may need to make small changes to programs so that they fit your data.

There are of course other reasons; such as you can soon learn to do simple things, which may be very useful. However, if you are going to change programs then you need to know about something we have not mentioned yet—Vectorisation. Matrix algebra can be exploited to avoid "do loops" (for and while statements) in programs. Sometimes this happens naturally, because one is programming a matrix expression in the first place. Sometimes it takes a little thought (and sometimes a lot of thought!) to realise that what looks like a series of calculations can be written as a single matrix expression. MATLAB calls this vectorisation. These "clever" bits of the program can be the hardest to decipher, but you need to learn to recognise it if you want to tinker with programs.

Lets start by looking at something that you have seen before. In the TDeious Fourier transformation program,[1] we needed to compute a matrix of cosines and a matrix of sines. Rather than use a for loop we set up some matrices and then did all the calculations in two simple lines of code.

```
n=size(y,1); % number of data
% points
k=n/2; % half number of data
% points
t=[1:n]'; % n x 1 col vector
% 1,2,...,n
p=[1:k]; % 1 x k row vector
% 1,2,...,k
wpt=(2*pi/n)*(t*p);
% n x k matrix with ij'th
% element 2(pi)ij/n
```

Now we are ready for the clever lines that demonstrate that it works with MATLAB functions.

```
C=cos(wpt); % n x k matrix of
% cosines
S=sin(wpt(:,1:(k-1)));
% n x k-1 matrix of sines
% (omit last one because it
% would be a col of zeros
% anyway)
```

Most MATLAB functions will work on vectors or matrices just as they do on scalars. Thus it is more efficient, both for ease of programming and speed of execution, to construct the matrix and then apply the sine and cosine functions.

Are you ready for something a little more challenging? In the last column[2] the LitmusB program contained the lines:

```
% Estimate absorption
% coefficients by a least
% squares regression
% THIS WILL BE EXPLAINED ON
% ANOTHER OCCASION
bcoefb=cb'*ab/(cb'*cb);
```

This computes the least-squares slope coefficients for 48 separate fits of straight lines through the origin, all in one go.

If we have n pairs of observations (y_i, x_i), then the formula for the slope coefficient is

$$b = \sum_{i=1}^{n} x_i y_i \Big/ \sum_{i=1}^{n} x_i^2$$

In our example, $n = 9$, and the 9 concentrations x_i are contained in the 9×1 vector **cb**. We have absorptions y_i for, and wish to compute regressions for, 48 different wavelengths. The matrix **ab** is 9×48, with the 9×1 vectors of y_i for different wavelengths making up its 48 columns.

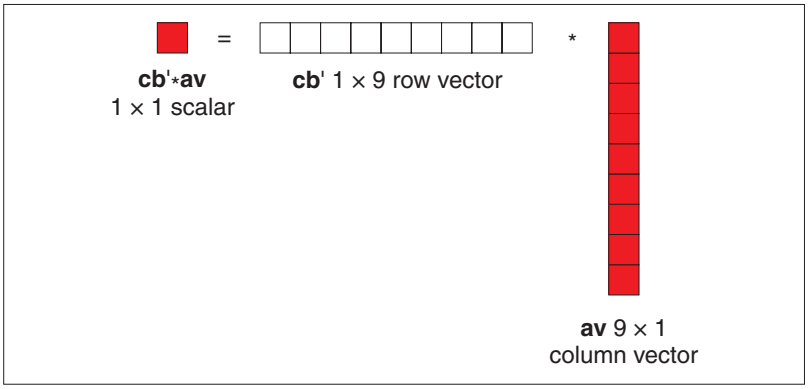

Figure 1. A scalar produced as the product of a row vector and a column vector.

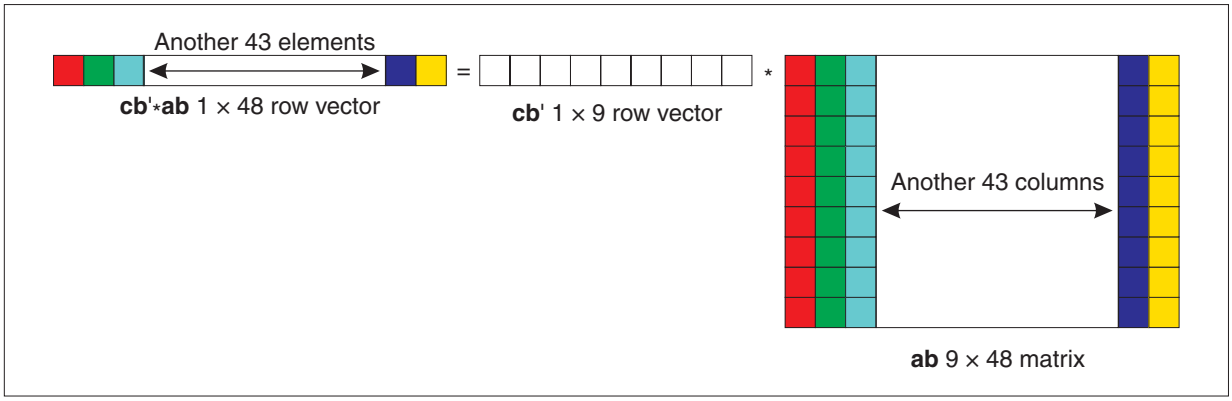

Figure 2. A row vector produced as the product of a row vector and a matrix.

Thinking one wavelength at a time, one could extract the appropriate column of **ab** into a 9 × 1 vector, **av** say. Then the slope b for that wavelength could be computed as

```
b = cb'*av/(cb'*cb);
```

Looking at the top of this, **cb′** is a 1 × 9 row vector with entries (x_1, …, x_9) whilst **av** is a 9 × 1 column vector with entries (y_1, …, y_9). Thus, with the usual rules for matrix multiplication, (Figure 1) the product **cb′*av** is a scalar whose value is the sum of products of x_i and y_i, i.e. the numerator in the equation for b. Similarly the product **cb′*cb** is another scalar, the sum of squares of the x_i, which is the denominator in the equation.

The point, though, is that we do not need to extract the columns of **ab** to do this. We can leave them in place and do all 48 computations at once. What makes it particularly easy to do this is the fact that the x_i are the same in each of the 48 cases. Thus the (scalar) divisor computed as **cb′*cb** is also the same for each of the 48 calculations, and this appears unchanged in the original line of code

```
bcoefb=cb'*ab/(cb'*cb);
```

What is different is the numerator, which as the product of the 1 × 9 row vector **cb′** and the 9 × 48 matrix **ab** is now a 1 × 48 row vector. The matrix multiplication rules (Figure 2) mean that the first element of this vector is the product of **cb′** and the first column of **ab**, the second element is the product of **cb′** and the second column of **ab**, and so on. Thus we have calculated the 48 numerators that we need for the 48 slopes without dismembering **ab**. Dividing by **cb′*cb** then gives us a vector of 48 slope coefficients.

This is neat and simple. It is also an obvious thing to do, because the absorptions naturally start in a matrix. We just have to realise that it is not necessary to extract them. Some examples are more artificial, in that one constructs a matrix from smaller parts especially so that vectorisation can be used. In trying to understand someone else's code, it can be helpful to draw a picture of the matrices involved, so that one can clearly see where the original quantities, e.g. the vectors of absorptions in this case, fit in.

Examples

The best way of learning something is to try to do it yourself! So here are some examples for you to do. The answers will be found on the *Spectroscopy Europe* website at: http://www.spectroscopyeurope.com/td_col.html.

1) Convert a vector of Fahrenheit temperatures to Centigrade

```
% Program f2c converts
% degrees F to degrees C.
Tf=[0:20:240]; % vector of
% temperatures from 0 to 240
% at intervals of 20.
format short % restricts
% output to four decimal
% places.
n=size(Tf,2); % Check size of
% array
TCc=zeros(1,n); % Set-up
% vector for results.
for k=1:n % Repeat for each
% observation in the input
% vector.
TCc(1,k)=(Tf(1,k)-32)*5/9;
% Calculate the converted
% temperature.
end
TCc % Print the answers
```

2) Centring data

We normally centre data before principal component analysis (PCA). That is we find the mean value of the spectral data at each point in the spectrum; then we subtract this mean for each observation. Once you have the spectral data this can be done with one line of code. Your task is to replace the "For loops" with one line.

```
load('S12a'); % S12a is
% spectral data
X = spec; % Array of 12
% spectra containing 864 data
% points;
% This is an array of 12 x
% 864
% Centre data
% For loop to centre data
n=size(X,1) % number of
% spectra
j=size(X,2) % number of
% readings in spectrum
Xd=zeros(n,j); % An array to
% hold the answers
Xda=zeros(n,1); % An array to
% hold the answers for a
% column
for k=1:j % Repeat for each
% data point in the spectrum
Xs=X(:,k); % Column of X-data
mn=sum(Xs)/n; % average of
% column

for m=1:n % Repeat for each
% observation in a column of
% data
Xda(m)=Xs(m)-mn; % Subtract
% average from each value in
% column
end

Xd(:,k)=Xda; % Put column
% into result array
end
```

3) MSC transformation of spectroscopic data

Multiple Scatter Correction (MSC)[3] is widely used in NIR spectroscopy for attempting to correct for scatter in dif-

fuse reflection spectra before using the data in calibration calculations. The method involves the calculation of a mean spectrum for a set of data (averaged in the wavelength dimension) and then calculates corrections for each spectrum relative to the mean spectrum. The spectral data for each spectrum is regressed against the mean spectrum and this provides an intercept, *a*, and slope, *b*, for each spectrum. The spectrum is then corrected by subtracting a from each observation and dividing the result by *b*: **x**(corr) = (**x** – *a*)/*b*; where **x** is the original spectrum and **x**(corr) is the corrected spectrum. You can read a full description of MSC and enjoy a sneak preview of a forthcoming book by going to:

http://www.nirpublications.com/userfriendly/.

Here is a program that does the calculations. Can you replace the "for loops" with appropriate matrix algebra?

```
% MSC for TD13.6

% multiplicative scatter
% correction

load('S12a'); % Spectral data
% 12 x 864 array
X=spec;
[r,c]=size(X);
%
% Replace the for loops
msp=zeros(1,c); % array for
% means
for k=1:c % repeat for each
% wavelength
msp(1,k)= sum(X(:,k))/r;
end
Gm=mean(msp); % the group
% mean
cmsp=zeros(1,c);
for k=1:c
cmsp(1,k)=msp(1,k)-Gm;
end
div=cmsp*cmsp'; % this is the
% sums of squares as above
B=(X*cmsp')/div; % This is a
% repeat of the bcoefb in the
% text above!
A=mean(X')'-B*Gm; % "mean" is
% a MATLAB function that
% calculates an average value
% of a column vector.
% Applied to a matrix,
% mean(X), returns a row
% vector of column means.
% Two final for loops or one
final line?
Xmsc=zeros(r,c);
for j=1:r
for k=1:c
Xmsc(j,k)=(X(j,k)-A(j))/B(j);
end
end
```

We hope that you do not need any knowledge that has not appeared in these columns but you may need to revise the earlier lessons, all of which are available from the *Spectroscopy Europe* web site:
http://www.spectroscopyeurope.com.

References

1. A.M.C. Davies and T. Fearn, *Spectrosc. Europe* **12(4),** 28 (2000).
2. A.M.C. Davies and T. Fearn, *Spectrosc. Europe* **13(4),** 22 (2001).
3. P. Geladi, D. McDougall and H. Martens, *Appl. Spectrosc.* **39,** 491 (1985).