Doing it faster and smarter (Lesson 6 of Matrix Algebra)

A. M.C. 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

Twenty years ago near infrared (NIR) spectroscopists were suddenly faced with an avalanche of data. Many of us had been using filter instruments which produced measurements at 19 wavelengths for every sample and we had been trying to learn how to use this data in regression analysis. With the arrival of commercial NIR scanning spectrometers we had 700 measurements per sample! What we wanted to do was to use our research spectrometer to find the few wavelengths that could be utilised by a filter instrument for practical application of NIR spectroscopy. The problems were partially one of methods (Harald Martens found the PLS solution and changed everything) but also of scale. Just to find the best pair of wavelengths from the possible 700 required nearly a quarter of a million calculations. Using our Nova 4 computer we estimated that it would take five days, if the computer would run that long without crashing! There had to be another approach. The solution we adopted in my group at the Institute of Food Research was a two-stage attack.[1,2] We first calculated a low-resolution set of results at 35 nm intervals and displayed them, after conversion to a colour scale, on a colour monitor. This required 630 calculations and was used to select manually up to 20 areas for high-resolution computation. Each of these areas required 820 calculations and 20 of them took around 16 hours of computing. It was my first collaboration with Tom Fearn who gave us a FORTRAN program he had written for computing "best pairs". Our system employed the Nova 4 computer for the computations and an Apple IIe computer for the display. It was the first colour monitor in the Institute and the wavelength pairs were found and selections were made using a cursor and a "games joy-stick" controlled by a program written in machine code by one of the group members—another first! The system was called CROME after the founder of the 19th century Norwich School of Painters.

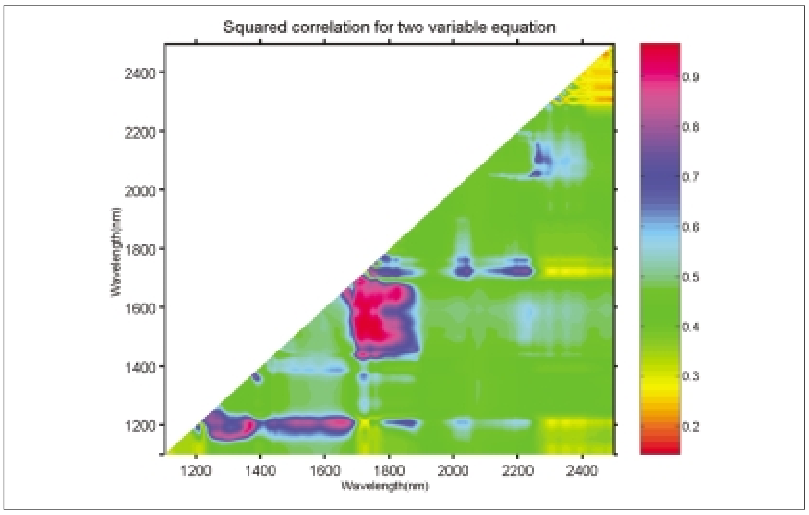

Figure 1. The "CROME" display of correlation coefficients for pairs of wavelengths.

The pictures produced by the system were interesting and the results were useful but not startling. The combination of new computers and new software resulted in the need to re-write the system and we never found the time and energy to do it. No commercial software has ever offered such a display and as far as I am aware, Karl Norris was the only NIR researcher to test the idea.

I wondered how much more efficient modern computers and matrix algebra would be in tackling the problem and offering everyone the opportunity for using this visualisation of NIR data—we all take colour monitors and mouse-controlled cursors for granted. I asked Tom Fearn if he would write a matrix algebra program to do the CROME calculations. He sent it to me a few days later and I was stunned by the result, Figure 1. The MATLAB program computed the complete set of 244,650 calculations and displayed the results in about five seconds. That is an improvement of 100,000 times faster than in 1982! All of the speed improvement is due to memory/CPU speed improvements. The original program would have been much more efficient than the new one. However, using matrices makes the program easier to write, and the efficiency with which MATLAB deals with the matrix calculations means that the penalty one pays for this ease of programming is not too high. The remainder of this article shows how it is done.

The calculations

If we have a matrix of NIR readings over 700 wavelengths ($x_1 - x_{700}$) for many (n) samples each with a value for the reference chemistry, y; then the correlation value, R^2, that we want to calculate is (for the first pair of x-values) given by

$$R^2 = \frac{r_1^2 + r_2^2 - 2r_1 \times r_2 \times r_x}{1 - r_x^2} \qquad (1)$$

where r_1 is the simple correlation between y and x_1, r_2 is the simple correlation between y and x_2 and r_x is the correlation between x_1 and x_2. The result, R^2, is the squared correlation for the two-variable equation predicting y from x_1 and x_2.

In order to do these calculations for all possible pairs we need to compute a matrix of simple correlations then we can use them to do the remaining computations.

It is not too surprising that MATLAB has a function to compute correlations so we use it to do the hard work and then just do a few (!) more calculations. It is more convenient to write the program as an organising program that calls a function where all the calculations are done, rather than a single program. So here are the first few lines of the main program which opens the data files. The data is a set of 40 samples of biscuit dough of known fat content which have been scanned over the range 1100–2498 nm. There is an error in sample 23 so it is excluded from the data. As usual the MATLAB code is shown in red and comments in green.

```
fid=fopen('nirc.asc','r');
dat=fscanf(fid,'%f',[p,n]);
status=fclose(fid);
dat=dat';
X=dat([1:22 24:40],:);
% 39 x 700 matrix of spectra,
% cal set, obs 23 out
fid=fopen('labc.asc','r');
dat=fscanf(fid,'%f',[q,n]);
status=fclose(fid);
dat=dat';
Y=dat([1:22 24:40],:);
% 39 x 4 matrix of lab
% values, cal set, obs 23 out
y=Y(:,1);
% 39 x 1 vector of fat values
wl=[1100:2:2498]';
R2=crome(X,y);
% This is the call to the
% function that does the
% calculations.
```

The data, an **X** matrix of 39 × 700 and a **y** vector of 39 values for fat content, have been sent to a function named "crome". So now we switch our attention to this function. The first few lines of instructions can be explained reasonably easily:

```
function R2=crome(X,y);
% returns matrix of squared
% correlations for regression
% on pairs
% input
% X - n x p matrix of
%      x-variables
% y - n x 1 vector with the
%      y-variable
%
% Get no of variables
p=size(X,2);
% Find all the pairwise
% correlations.
% The MATLAB function
% corrcoef returns all the
% pairwise correlations for
% the variables in the
% columns of the matrix that
% is its argument. By joining
% X and y together as [X y]
% we get a (p+1)x(p+1) matrix
% C of correlation
% coefficients, where the top
% left p x p matrix has the
% pairwise correlations of
% the variables in X, the
% final column and the final
% row have the pairwise
% correlations of each x with
% y (and the bottom right
% hand element is 1, like all
% the diagonal entries).
C=corrcoef([X y]);
% Extract the p x p
% correlation matrix of the
% columns of X at the same
% time setting the diagonal
% to zero (rather than 1)
rx=C(1:p,1:p)-eye(p);
% correlation matrix of
% 700x700 x-vars, with 0's on
% the main diagonal (remember
% that "eye" is the "identity
% matrix" with 1's on the
% main diagonal and zeros in
% all other elements).
%
% Extract the vector of
% correlations of x with y
rxy=C(1:p,p+1);
% vector of simple
% correlations of y with each
% x
%
% Compute R-squared, R2,
% for all two-var eqns
% note: the fiddling around
% on the diagonal avoids 0/0
% in equation 1 when
% regressing on the same
% variable twice. R-squared
% is the square of the simple
% correlation in this case.
rxy2=(rxy.*rxy)*ones(1,p);
% This is a simple
% multiplication of two
% vectors to create a vector
% of rxy2, followed by a
% matrix multiplication with
% a vector of 1x700 1's to
% produce a 700x700 matrix of
% r2, with 700 identical
% columns.
div=ones(p,p)-rx.*rx+eye(p);
% div is the divisor in
% equation [1]. It is
% composed of a 700x700
% matrix with 2's on the main
% diagonal and 1 minus the
```

Table 1. Values of *r* and *r*² for *xy*.

Wavelength x	r for xy	r^2 for xy
1100	–0.5554	0.3085
1102	–0.5557	0.3088
1104	–0.5567	0.3099
1106	–0.5575	0.3108
1108	–0.5586	0.3121

```
% squared x-variable
% correlations off-diagonal.
R2=(rxy2+rxy2'- ...
2*(rxy*rxy').*rx)./div;
% If this was a description
% of a chess match this move
% would be marked with a ! or
% !!
```

How can $(r_{xy}^2 + r_{xy}^2$¢$)$ be equivalent to $(r_1^2 + r_2^2)$? In order to explain this move (line) we need to leave the program and look at a very small example. If we only use the first five wavelengths then we can get the program to print some intermediate results.

r_{xy} is:

```
–0.5554
–0.5557
–0.5567
–0.5575
–0.5586
```

These have been written in Table 1 with the wavelengths identified and with the corresponding squared value. However, in the program we wrote them into the columns of a 5 × 5 matrix:

r_{xy}^2 is:

```
0.3085 0.3085 0.3085 0.3085 0.3085
0.3088 0.3088 0.3088 0.3088 0.3088
0.3099 0.3099 0.3099 0.3099 0.3099
0.3108 0.3108 0.3108 0.3108 0.3108
0.3121 0.3121 0.3121 0.3121 0.3121
```

and the transpose r_{xy}^2¢ is:

```
0.3085 0.3088 0.3099 0.3108 0.3121
0.3085 0.3088 0.3099 0.3108 0.3121
0.3085 0.3088 0.3099 0.3108 0.3121
0.3085 0.3088 0.3099 0.3108 0.3121
0.3085 0.3088 0.3099 0.3108 0.3121
```

What we do in the clever move is to utilise the symmetry of the operation (and we use the same trick twice). If we imagine these matrices in a vertical stack, see Figure 2, and then think about the data that we need for a particular computation, say R^2 for 1104 and 1100. The r_{1104} value is 0.3099 and the r_{1100} value is 0.3085. We could find these in the column of r_{xy}^2 values in Table 1, but if we look vertical up

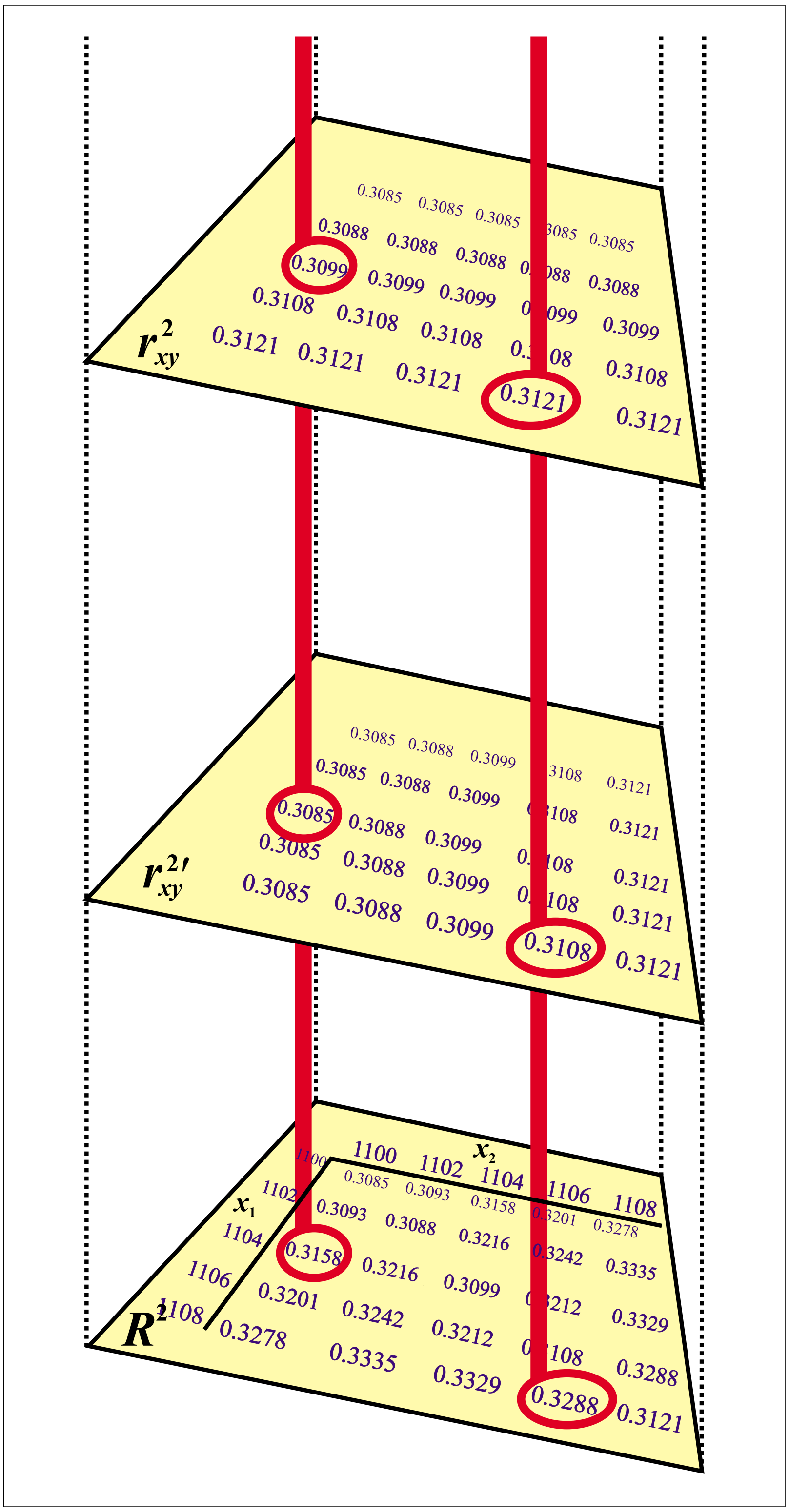

Figure 2. Representation of the use of r^2_{xy} and r^2_{xy}¢ matrices in the calculation of a matrix of R^2 results. Note: the wavelengths have been added to the R^2 matrix to aid comprehension.

from where the result will be written these two values will be found in the r^2_{xy} and r^2_{xy}¢ matrices. The same is true for any calculations. R^2 for 1108 and 1106 requires 0.3121 and 0.3108. It is almost like magic (but you were warned about the magical properties[3] of matrices in the first lesson)!

Now we can return to the program with our values of R^2, (in fact we return to the main program). There is just one more thing to explain. You may have noticed that we have worked with square matrices and in fact we have calculated 490,000 R^2s rather than the 244,650 that we actually need (r_1 and r_2 give the same result as r_2 and r_1). The results we want are in the triangle and we also do not need the diagonal (on which $r_1 = r_2$) so we just forget them by setting them to the code NaN (not a number). Then all we do is to call up the correct MATLAB display function and the rest of the hard work is done for us.

```
% construct triangular matrix
% (includes diagonal) of
% 'NaN's
M=ones(size(R2))*NaN;
Mt=tril(M);
% use triu here if you'd
% rather delete the other
% half
%
% Add to R2, which replaces
% the whole triangle by NaN's
% which don't get plotted
R2t=R2+Mt;
h1=figure;
pcolor(wl,wl,R2t)
shading('flat')
colormap('hsv')
xlabel('Wavelength(nm)')
ylabel('Wavelength(nm)')
title('Squared correlation
for two variable equation')
set(gca,'TickDir','out')
axis([1100 2498 1100 2498])
```

This example shows how being very inefficient with storage space allows the use of efficient computation. We have quite a lot of space in modern desktop computers so we might as well use it and matrix algebra is a very good method for achieving this aim. This is certainly not the fastest result possible; C code would be faster but it is much more difficult to write. It is not necessarily easy to see how to design a program but drawing pictures is the clue.

Another question you might ask is "Is the plot any use?" Well with this plot you can very easily find a pair of wavelengths that will give you a correlation of 0.96 which is quite good for this data set. You know that you are not missing any important areas and the plot is reassuring to spectroscopists who know that fat is associated with CH vibrations and absorptions due to CH occur around 1200, 1725 and 2310 nm.

References

1. A.M.C. Davies, M.G. Gee and P.W. Foster, *Lab. Practice* **33(5),** 78 (1984).
2. A.M.C. Davies and M.G. Gee, *Lab. Practice* **34(7),** 36 (1985).
3. A.M.C. Davies, *Spectroscopy Europe* **12(2),** 24 (2000).