The TDeious way of doing Fourier transformation (Lesson 2 of matrix algebra)

A.M.C. Davies

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

Tom Fearn

Department of Statistical Sciences, University College London, Gower Street, London, UK

At the end of the last column[1] we promised that this time we would show how matrix algebra can be used for real computational tasks. The chosen task is Fourier transformation (FT) of a near infrared (NIR) spectrum. Those who know Tony Davies will not be surprised at this choice of subject but in the third lesson the reason for wanting to do the obvious will become apparent. Many of you will know and use something called the fast Fourier transform (FFT)[2] but we want you to understand exactly what is happening during FT and we will do this without the added complication of the FFT. If anyone would like an electronic version of the MATLAB code below, you can find it on the *Spectroscopy Europe* web site, at http://www.spectroscopyeurope.com/td_col.html. We should stress though that this is not an efficient way of computing a Fourier transform—the FFT would be very much faster.

We reminded you quite recently[3] of the basis of FT; that any curve can be approximated by the summation of a series of sine and cosine waves. The idea of using FT for the data compression of NIR spectra was described by Fred McClure and his colleagues and we will use their procedure,[4] as it avoids becoming involved with complex numbers.

If we have a spectrum, which has been recorded at n equally spaced wavelength intervals, $y(1)$, $y(2)$, $y(3)$, ..., $y(n)$ then it can be approximated by:

$$y(h) = a_0 + \sum_{j=1}^{k} a_j \cos\left(\frac{2\pi jh}{n}\right) + \sum_{j=1}^{k-1} b_j \sin\left(\frac{2\pi jh}{n}\right) \qquad (1)$$

where the coefficients are given by

$$a_0 = \frac{1}{n}\sum_{h=1}^{n} y(h)$$

$$a_j = \frac{2}{n}\sum_{h=1}^{n} y(h)\cos\left(\frac{2\pi jh}{n}\right)$$

for $j = 1, 2, 3, \ldots, k-1$

$$a_k = \frac{1}{n}\sum_{h=1}^{n} y(h)\cos \pi h$$

and

$$b_j = \frac{2}{n}\sum_{h=1}^{n} y(h)\sin\left(\frac{2\pi jh}{n}\right)$$

for $j = 1, 2, 3, \ldots, k-1$.

In the previous article we put instructions directly into the MATLAB program but when we have more complex requirements it is normal practice to enter the instructions into a file and then instruct MATLAB to execute the file. So now we will give you the program (instructions are in red while comments are in green) and explain it as it would proceed.

```
% TDeious Fourier Transformation
% Number of pairs of Fourier
% coefficients to use in reconstruction
npairs=20
fcs=npairs+1
```

While the primary use of FT is to reorganise information (as in converting an IR interferogram to a spectrum), another very important application is in data compression. If we start with n spectral observations we will have $n/2$ pairs of Fourier coefficients but many of the high frequency coefficients will be close to zero and they can be ignored with very little loss of information.

```
% Get some data
load testspec;
```

Our test spectrum is an NIR spectrum of a piece of a PET polymer, which was recorded at 2 nm intervals over the range 1100–2498 nm.

```
d=H_XV'; % d is a 700 x 1 column vector
```

The data had been named "H_XV" but it is more convenient for us to rename it to the single character "d". H_XV was a 1×700 row vector which we transposed to make d a 700×1 column vector.

```
dp=size(d,1) % dp is 1 x 1; known as a
% "scalar"
```

"size" is a MATLAB function which will find the dimensions of a matrix.

Note that the statement is not terminated by a ";" so the value of dp will be printed on the screen.

```
w=[1100:2:2498]'; % w is a 700 x 1
% column vector
```

"w" is the wavelength scale for plotting; 700 wavelengths have been calculated from 1100 to 2498 nm in 2 nm intervals. These correspond to the wavelengths of the original spectrum.

```
% use this spectrum for FT
% calculates FT using equations from
% McClure's paper in Appl. Spectrosc.
```


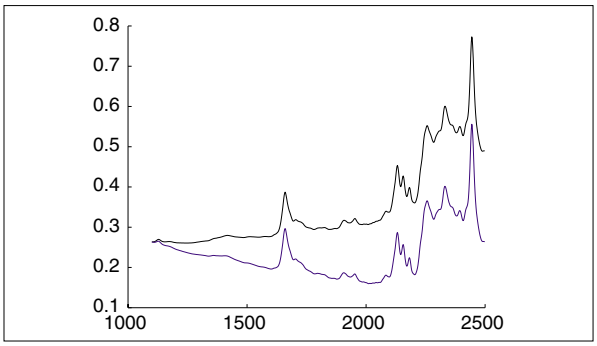

Figure 1. NIR spectrum of a sample of PET; original spectrum (black) and titled spectrum (blue); plotted as log1/*R* against wavelength, nm.

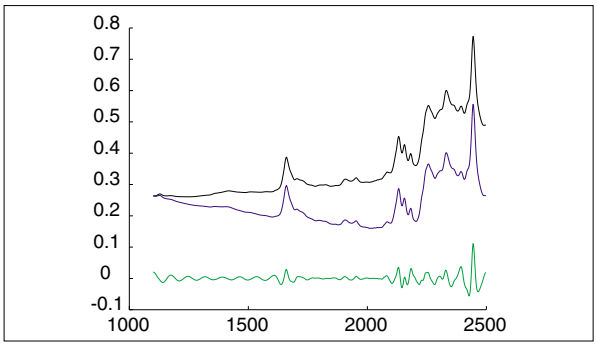

Figure 2. The original (black), tilted (blue) and the difference spectra (green) using 20 pairs of Fourier coefficients for the reconstruction of the spectrum.

```
hold on
```

Allows us to plot several lines on the same figure.

```
plot(w,d,'k');
```

Plots the original data; "k" specifies a black line

```
% Tilt spectrum to make ends equal
```

NIR spectra usually show an upward linear trend. One of the assumptions of using FT is that the waveform repeats to infinity, from each end. If the ends of the spectrum are not equal this discontinuity will cause "ringing". We make them equal by joining the ends of the spectrum with a straight line and then subtracting it across the spectrum.

```
% calculate slope of the line
dm=dp-1; % dm is 1 x 1
s=(d(dp)-d(1))/dm; % s is 1 x 1
e=s*[0:dm]'; % e is 700 x 1
```

Our first matrix operation. We have just calculated the correction for our 700 point data and then with another line we can do 700 subtractions! s is 1/699 of the difference between the heights of the first and last spectral points. The next piece of code subtracts 0 from the first point, s from the second, 2s from the third and so on.

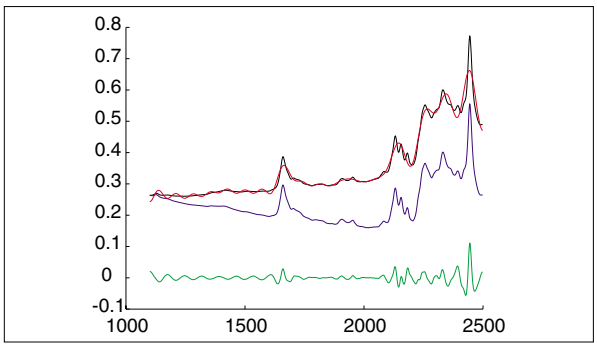

Figure 3. The original (black), reconstructed (red), tilted (blue) and difference spectra using 20 pairs of Fourier coefficients for the reconstruction of the spectrum.

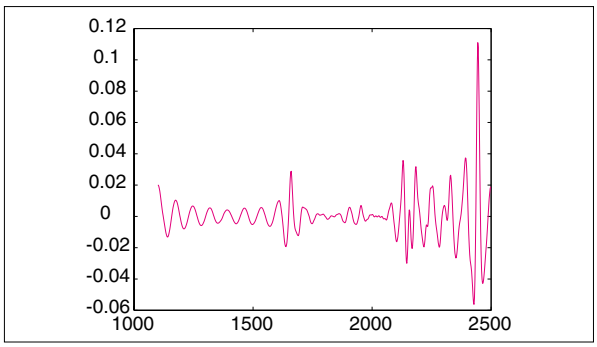

Figure 4. The difference spectrum plotted on its own scale using 20 pairs of Fourier coefficients for the reconstruction of the spectrum.

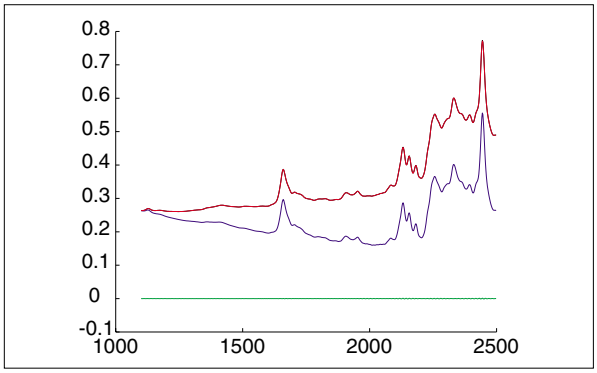

Figure 5. As Figure 3 using 100 pairs of Fourier coefficients; the original spectrum (black) is almost completely covered by the reconstructed spectrum (red).

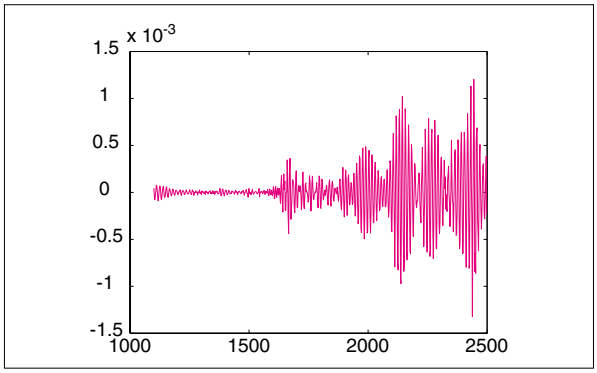

Figure 6. As Figure 4 using 100 pairs of Fourier coefficients; note that the log1/*R* scale is $\times 10^{-3}$.

```
% adjust data by slope of line
e1=d-e; % e1 is 700 x 1
plot(w,e1,'b');
pause
```

We plot the corrected spectrum and the program waits for an input before continuing. "b" specifies a blue line. The picture at this point is shown in Figure 1.

Now we are ready to do the FT. First we set up some matrices and constants:

```
y=e1; % this is the tilted spectrum,
% 700 x 1
n=size(y,1); % number of data points
% (700)
k=n/2; % half number of data points
% (350)
t=[1:n]'; % n x 1 column vector
% 1,2,...,n
p=[1:k]; % 1 x k row vector 1,2,...,k
wpt=(2*pi/n)*(t*p); % wpt is an n x k
% matrix
```

t is an $n \times 1$ column vector and p is a $1 \times k$ row vector, so t*p is an $n \times k$ matrix with the element in the hth row and jth column equal to the hth element of t (which has been set equal to h) times the jth element of p, which is equal to j, resulting in jh. Thus the elements of wpt are $2\pi jh/n$. Now we find the sines and cosines of all these, leaving out the last sine, because it would give a column of zeros.

```
C=cos(wpt); % n x k matrix of cosines
S=sin(wpt(:,1:(k-1))); % n x k-1 matrix
% of sines
```

Since sin(matrix) replaces each element of the matrix with its sine, we were able to calculate 700×350 sines with a single command, no loops needed. Now we put the sines and cosines together in one big matrix, with a column of ones at the start.

```
X=[ones(n,1) C S]; % put together into
% n x n matrix
```

$\mathbf{X}$ is $n \times n$ (700×700) because it is made up of a 700×1 column of ones, a 700×350 matrix of cosines and a 700×349 matrix of sines, placed side by side. There is one column for each of the terms in the sum in Equation 1, and one row for each wavelength.

If we were to write Equation 1 in matrix form as $\hat{y} = \mathbf{X}a$, where $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_n)'$ and $a = (a_0, a_1, \ldots, a_k, b_1, \ldots, b_{k-1})'$ then the $\mathbf{X}$ just constructed is the one that would be required. To get $\hat{y}(h)$ we multiply the hth row of $\mathbf{X}$, which contains $\{1, \cos(2\pi 1h/n), \ldots, \cos(2\pi kh/n), \sin(2\pi 1h/n), \ldots, \sin[2\pi(k-1)h/n]\}$ by the column vector a, giving exactly the sum in Equation 1. Interestingly, the same $\mathbf{X}$ will give us the coefficients a_j and b_j as given by the formulae below Equation 1, using the following code

```
div=n*[1 (1/2)*ones(1,k-1) 1
(1/2)*ones(1,k-1)]'; % div is n x 1
% vector
fc=(X'*y)./div; % fc is n x 1 vector of
% Fourier coefficients
```

The first of these two lines sets up a vector containing n, $n/2$ repeated $k-1$ times, n, and $n/2$ repeated $k-1$ times, which corresponds to the divisors in the equations for the coefficients. In the second line $\mathbf{X}'*y$ produces an $n \times 1$ (700×1) vector with the summations needed for the coefficients (the matrix product doing the summations for us) and then ./div divides element-by-element by the divisors we

have set up in div. The operators .* and ./ are useful. If matrices **A** and **B** have the same size then **A** .* **B** is the matrix we get by multiplying each element of **A** by the corresponding element of **B**, much simpler than the full matrix product **A***B. Similarly ./ results in element-by-element division.

The FT is complete! One remark worth making is that the coefficients we just calculated are actually the result of a least squares fit. Those who remember the formula $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'y$ for a least squares fit (all of you I'm sure) might wonder where the first bit of this equation has gone. In fact $\mathbf{X}'\mathbf{X}$ is a diagonal matrix, with the elements of div on the diagonal, so the expression simplifies. The standard formula would give the same result, after wasting an imperceptible amount of time multiplying two 700×700 matrices together and inverting the result.

```
% invert the transform
% make unrequired coeffi-
cients 0
fc(fcs:k)=0;
```

These are the high order *a* coefficients

```
fc(k+fcs:n)=0;
```

These are the high order *b* coefficients

```
xfit=X*fc; % matrix is 700
x 1
```

"xfit" is the reconstructed spectrum using the specified number of pairs of Fourier coefficients.

```
% Un-tilt the spectrum

xfits=xfit'+e'; % matrix
is 1 x 700 row % vector
```

We just add what we subtracted from the original spectrum

```
ds=e1-xfit; % matrix is
700 x 1
```

"ds" is the difference spectrum to show how well we have reconstructed the spectrum.

```
plot(w,ds,'g');
```

We plot the difference spectrum and then the reconstructed spectrum on the same plot. Figures 2 and 3; "g" and "r" specifies the colours of the lines.

```
pause
plot(w,xfits,'r');
hold off
pause
```

Now we plot the difference spectrum on its own scale in a new plot; Figure 4.

```
figure
plot(w,ds,'m');
```

For this first example we deliberately chose a small number of Fourier coefficients. A more appropriate choice would be 100. This produces the plots shown in Figures 5 and 6. The errors are small but we have achieved a 70% reduction in the size of the data.

In the third part we will make use of our Fourier transform in a novel way, which will introduce some further wonders of matrix algebra.

References

1. A.M.C. Davies, *Spectrosc. Europe* **12(2),** 24 (2000).
2. J.W. Cooley and J.W. Tukey, *Maths. Comput.* **19,** 297 (1965).
3. A.M.C. Davies, *Spectrosc. Europe* **11(6),** 24 (1999).
4. F.G. Giesbrecht, W.F. McClure and A. Hamid, *Appl. Spectrosc.* **35,** 210 (1981).