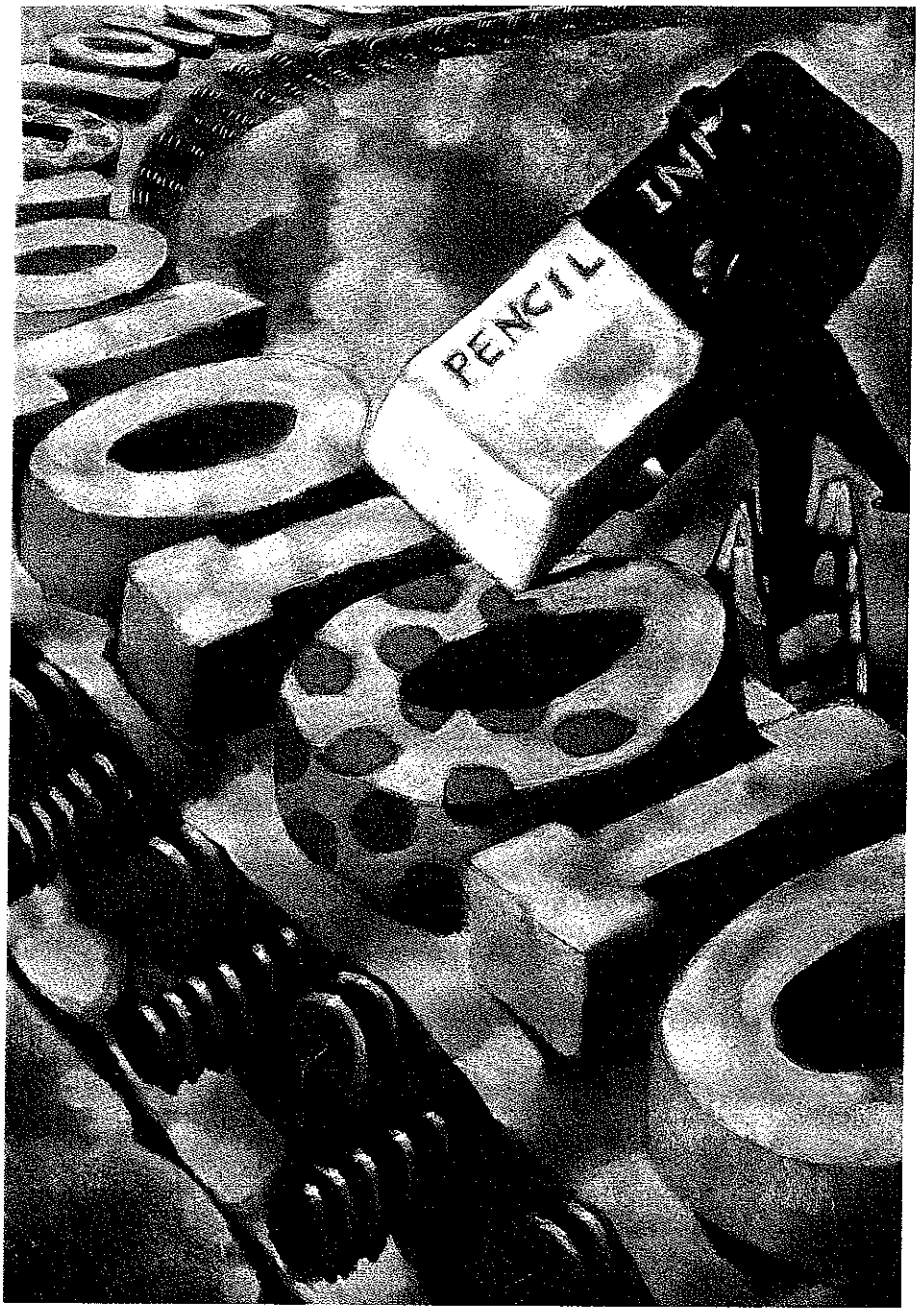


Running Fourier transforms take the bind out of examining spectra in a run of digital samples. This makes the technique particularly useful for designing digital filters, as Allen Brown explains.

Running Fourier transforms

The Fourier transform has become a well established method for determining the frequency content of signals. There are several variations of the Fourier transform and its implementation, the most common of which is the fast Fourier transform, or FFT.

One particular variation is the running Fourier transform. It has a number of interesting applications, including real-time multi-bandpass digital filtering.

When provided with a batch of digital samples, applying the FFT gives the complex spectral set. If there are N data samples in the original data set, you are required to perform of the order of, $N\log_2(N)$ calculations when using the FFT. Given a second set of N data samples you would be expected to perform another $N\log_2(N)$ calculations.

What if you had a continual stream of data samples and you wanted to calculate a new spectrum every time a new sample arrives? Would you have to perform $N\log_2(N)$ calculations each time?

Using the running Fourier transform makes this unnecessary. In fact the number of calculations is greatly reduced. If you have calculated the m^{th} spectra, the $(m+1)^{th}$ spectra, after the arrival of the new sample $x(N+m)$, using the running Fourier transform is given by,

$$X_{m+1}(k) = e^{j2\pi\frac{k}{N}}\left\{X_m(k) + \frac{x(N+m) - x(m)}{N}\right\} \quad 1$$

where k is the frequency index, (k=0,1, 2... N/2–1). The derivation of this expression is given in the panel entitled 'Deriving...'.

As you can see from this expression, once you have calculated the first spectrum $\{X_o(k)\}$ successive spectra are calculated recursively. The spectrum $\{X_m(k)\}$ has N/2 complex values. But in your application you may only be interested in the behaviour of the spectrum at particular frequencies. As a result, you only need to calculate the spectral values of interest.

You can see from equation 1 that you need to remove the oldest data point $x(m)$ and add the newest data point $x(N+m)$. This means that you have to store all the data points N in the current batch. This is best achieved by using a *circular buffer*, which I discuss later.

Monitoring discrete spectral values

To illustrate how the running Fourier transform operates, consider this example.

List 1. Example of how the counter is implemented in C. The percentage sign represents modulus.

```
x[(0+7)%7]=x[0], x[(1+7)%7]=x[1], ...
        x[(6+7)%7]=x[6], x[(7+7)%7]=x[0], x[(8+7)%7]=x[1], ...
```

List 2. Implementing the running Fourier transform counter example in C. Two-dimensional arrays X[0][k] and X[1][k] represent the real and imaginary spectral values.

```
NEW_X[0][k] = OLD_X[0][k]*cos(ak)
    -OLD_X[1][k]*sin(ak)-recipN*(x[newest]-x[oldest])*cos(ak) ;

NEW_X[1][k] = OLD_X[0][k]*sin(ak)
    +OLD_X[1][k]*cos(ak) + recipN*(x[newest]-x[oldest])*sin(ak);

MAG_X[k]=sqr(NEW_X[0][1]*NEW_X[0][1]
    +NEW_X[1][k]*NEW_X[1][k]);
```

A signal is being sampled at 20kHz – i.e. with a sample period of 50µs. New spectral values are required for every new sample produced. But let's say that we are only interested in the frequency components at 3.35kHz, 4.050kHz and 6.650kHz.

If there are 200 real data samples in a batch, so that N becomes 200, the spectrum will comprise 100 unique complex values. The time over which the samples are collected will be 200×50µs=10ms. The spectral resolution for each k value will be 1/10ms, or 100Hz.

This makes the integer values of the index k for the above frequencies 3.350/100=33, 4000/100=40 and 6600/100=66. Out of the 100 values in the spectral set we would only calculate,

$\{X_m(33), X_m(40), X_m(66)\}$.

Since 1/200=0.005, by using equation 1, these become,

$$X_{m+1}(33) = e^{j1.036}\left\{X_m(33) + 0.005\left[x_{newest} - x_{oldest}\right]\right\} \quad 2$$

$$X_{m+1}(40) = e^{j1.256}\left\{X_m(40) + 0.005\left[x_{newest} - x_{oldest}\right]\right\} \quad 3$$

$$X_{m+1}(66) = e^{j2.073}\left\{X_m(66) + 0.005\left[x_{newest} - x_{oldest}\right]\right\} \quad 4$$

To start the calculation it's necessary to calculate $\{X_0(33), X_0(40), X_0(66)\}$. This can be achieved by performing a discrete Fourier transform for the three values. The 200 data samples collected, labelled $x(0)$ through to $x(199)$, are subjected to the discrete Fourier transform, which is defined as,

$$X_0(k) = \frac{1}{N}\sum_{n=0}^{n=N-1} x(n)e^{-j2\pi nk/N} \quad 5$$

but out of the spectrum, in this example, we only want three spectral values so,

$$X_0(33) = 0.005\sum_{n=0}^{n=199} x(n)e^{-j1.036n} \quad 6$$

$$X_0(40) = 0.005\sum_{n=0}^{n=199} x(n)e^{-j1.256n} \quad 7$$

$$X_0(66) = 0.005\sum_{n=0}^{n=199} x(n)e^{-j2.073n} \quad 8$$

Incidentally, you can extend this argument for as many frequency components as you like. It's also worth noting that since the spectral values are complex they carry magnitude and phase information.

Alternatively, each $X_m(k)$ may carry two channels of information since each is a quadrature pair – i.e. a sine and a cosine. The running Fourier transform is in effect filtering the required frequencies. In this example it is operating like a bandpass filter where the three spectral components are be fed into separate channels.

To demonstrate this working, the panel entitled 'A Mathcad model...' shows the running Fourier transform operating on a white noise signal and producing the outputs at the three frequencies. White noise contains components at all frequencies and the effect of the running transform is to isolate the real and imaginary components for the three frequencies cited in this example.

Circular buffering

As mentioned above, the best way of storing N data values is to use a

Deriving the expression for calculating the arrival of a new running Fourier transform sample

The discrete Fourier transform of a batch of N samples, labelled with the index m is,

$$X_m(k) = \frac{1}{N}\left\{\sum_{n=m}^{n=N+m-1} x(n)e^{-j2\pi(n-m)k/N}\right\} \quad A1$$

As a new sample arrives we want to recalculate the DFT for the batch labelled m+1,

$$X_{m+1}(k) = \frac{1}{N}\left\{\sum_{n=m+1}^{n=N+m} x(n)e^{-j2\pi(n-m-1)k/N}\right\} \quad A2$$

Extending the sum to include the sample for $n=m$ in the previous equation gives,

$$X_{m+1}(k) = \frac{1}{N}\left\{\sum_{n=m}^{n=N+m} x(n)e^{-j2\pi(n-m-1)k/N} - x(m)e^{j2\pi k/N}\right\} \quad A3$$

Removing the last term from the sum which involves $x(N+m)$ gives,

$$X_{m+1}(k) = \frac{1}{N}\left\{\sum_{n=m}^{n=N+m-1} x(n)e^{-j2\pi(n-m-1)k/N} + x(N+m)e^{-j2\pi N/N} - x(m)e^{j2\pi k/N}\right\} \quad A4$$

Since $e^{-j2pk}=1$ for integer values of k, equation A4 reduces to,

$$X_{m+1}(k) = e^{j2\pi k/N}\left\{\frac{1}{N}\sum_{n=m}^{n=N+m-1} x(n)e^{-j2\pi(n-m)k/N} + \frac{[x(N+m) - x(m)]}{N}\right\} \quad A5$$

By substituting A1 into A5 we obtain,

$$X_{m+1}(k) = e^{j2\pi k/N}\left\{X_m(k) + \frac{[x(N+m) - x(m)]}{N}\right\} \quad A6$$

which is the required result.

Digital frequency tracking

One possible application for the running Fourier transform is the design of a digital frequency tracker. Output from such a device would be a value – or voltage – proportional to the fundamental frequency of the input signal. By calculating the spectral values $X_{m+1}(K)$ for all the k values, the fundamental frequency will correspond to the maximum spectral value. Finding the maximum spectral value can be achieved by using a for-loop in your C program:

```
float max;
int freq_of_max;
max=0;
freq_of_max=0;
...
for (k=0;k<=N/2;k++)
{
      if(max<MAG_X[k])
      {
            max=MAG_X[k];
            freq_of_max=k;
      }
}
```

On completion of this 'for' loop, variable *max* will the magnitude of the maximum spectral point and *freq_of_max* will be its frequency. These can be used to feed a graphical display or even in some designs as part of a feed-back mechanism to stabilise a frequency at source, such as that of a rotating machine.

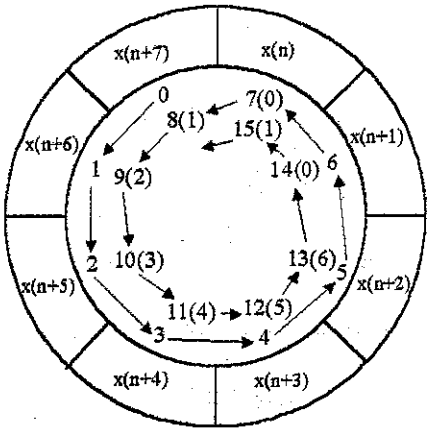


Fig. 1. The best way of storing N data values is to use a circular buffer where the newest data value replaces the oldest data value.

circular buffer where the newest data value replaces the oldest data value. **Figure 1** illustrates how a circular buffer operates, $x(n)$ is the oldest sample and $x(n+6)$ is the most recent.

The key aspect of a circular buffer is the address mechanism which is derived from a prime number. In this example, the counter counts from 0 to 6 and repeats. When there are seven addresses and the counter expires, it automatically points to the oldest address where the newest data sample $x(n+7)$ will be stored, assuming that the address counter is zero.

In this example the counter is generated by using,

(n+7) mod 7

which is defined as the remainder integer after $(n+7)$ has been divided by 7 which gives 7 different address numbers before the repeat begins. In order for this to work the divisor must be a prime number. To program this in C, you would use the percentage symbol to indicate modulo operation. **List 1** is an example that would effect a circular counter.

Mathcad model of the running Fourier transform

Define the indices used in the modelling.

Total number of data points $n := 0 .. 800$

The number of data points in a batch, $N := 200$

Number of batches $M := 500$

Batch index $m := 1 .. M$

Number of spectral values in spectrum $k := 0 .. \frac{N}{2}$

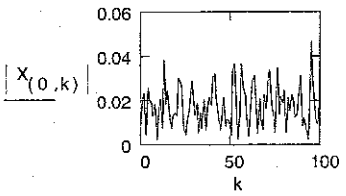
Generate 800 data points of random noise.

$x_n := (rnd(1) - 0.5)$

Derive the spectrum, using the DFT, from the first batch of N data points,

$$X_{(0,k)} := \frac{1}{N} \cdot \sum_{s=0}^{N-1} x_s \cdot e^{\frac{-2j\,p\,k\,s}{N}}$$

The spectrum of the first batch.



Having calculated $X_0(k)$, we go on to calculate the r-FT, $X_{m+1}(k)$ for m=0..M and k=33, 40 and 66 as examples.

$X_{m+1}(33)$, Eq:2

$$X_{(m+1,33)} := e^{1.36j}\left[X_{m,33} + 0.005\left(x_{N+m} - x_m\right)\right]$$

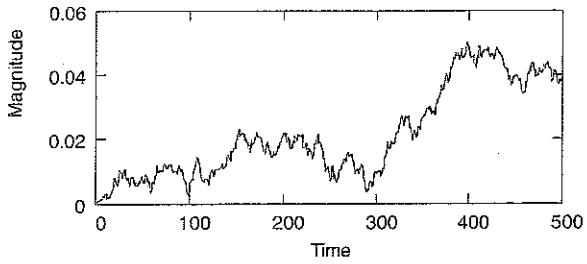
$X_{m+1}(40)$, Eq:3

$$X_{(m+1,40)} := e^{1.256j}\left[X_{m,40} + 0.005\left(x_{N+m} - x_m\right)\right]$$

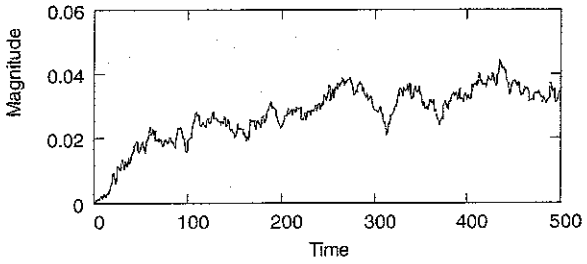
$X_{m+1}(66)$, Eq:4

$$X_{(m+1,66)} := e^{2.073j}\left[X_{m,66} + 0.005\left(x_{N+m} - x_m\right)\right]$$

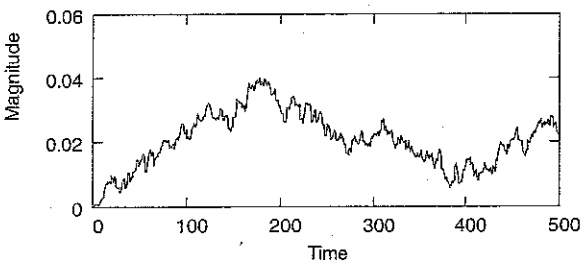
Plot of the growth of $|X_{m+1}(33)|$ vs time.



Plot of the growth of $|X_{m+1}(40)|$ vs time,



Plot of the growth of $|X_{m+1}(66)|$ vs time.



These plots demonstrate how the three spectral values vary as every new data point enters the batch and the new spectrum is calculated.

Implementing running transforms

The running Fourier transform is a very effective means for isolating frequency components from a signal. It is relatively straightforward to implement and code in C.

Since the spectral values are complex, you can use a two-dimensional array to define them, X[0][k] for the real values and X[1][k] for the imaginary values.

Taking equation 1 and equating real and imaginary components you can arrive at **List 2**. Here, $\alpha=2\pi/N$ and recipN=1/N (performing multiplications is quicker than divisions).

These two expressions can be coded directly within a loop and will perform a running Fourier transform operation. If the input data {x} is complex, the expressions can be modified to account for this.

Another attractive feature of the running Fourier transform is its ability to be implemented in real-time on a digital signal processor. You will be faced with the usual problems of having to evaluate the trigonometric functions sine and cosine, but you can use a look-up table or a reduced series expansion.

In the analysis presented here, no mention has been made of rounding effects. Equation 1 is recursive – involving new values derived from old ones. When implemented on a fixed-point processor it would probably show signs of accumulated rounding off.

A means of minimising this would be to perform a discrete Fourier transform for every few hundred thousand data samples or use a floating point processor. Several such processors have sine look-up tables and a circular addressing mode; examples are the Motorola *DSP96001* and the Texas Instruments *TMS320C30*. ■