Exstrom Laboratories LLC

P.O. Box 7651
Longmont, Colorado 80501
http://www.exstrom.com

A C Program For Spectrum Magnification: When An FFT Is Not Enough

by
Stefan Hollos, stefan@exstrom.com
Richard Hollos, richard@exstrom.com
Exstrom Laboratories, Longmont, CO

There are many science and engineering applications that require an accurate frequency spectrum or Fourier transform of a signal. The Fourier transform of a sequence of samples of a signal is shown in equation 1. [1]

$$Q(\Omega) = \sum_{n=0}^{N-1} q[n] e^{-jn\Omega} \tag{1}$$

This equation is written in dimensionless form, where $\Omega = \omega T$, T is the sampling interval, and $q[n] = q(nT)$, the n^{th} sample of the signal $q(t)$. The equation also assumes that the signal has a finite duration, so that there are only a total of N contiguous nonzero samples. $Q(\Omega)$ is a periodic function of the continuous variable Ω, with a period equal to 2π. The usual procedure is to evaluate $Q(\Omega)$ at a set of N evenly spaced points in the interval $\Omega = [0, 2\pi]$. This procedure is called a discrete Fourier transform (DFT) and is usually carried out using an algorithm called the fast Fourier transform (FFT). The DFT gives you the Fourier transform at the points $\Omega = 2\pi k/N$ $(k = 0...N-1)$. In terms of real frequencies this gives you a resolution of $\omega = 2\pi/(NT)$ or $f = 1/(NT)$. For many applications, this

[1] A book that provides a good background for most aspects of digital signal processing is:

A. V. Oppenheim, R. W. Schafer, Discrete-Time Signal Processing, Prentice Hall, Englewood Cliffs, NJ, 1989.

resolution may be sufficient. There are some applications however where the location of a spectral peak has to be determined with great accuracy. In this case the resolution provided by a DFT may not be adequate. One way of increasing the resolution is to simply pad the samples with extra zeros. For example to double the frequency resolution, you could add N zeros to the end of the $q[n]$ sequence, to give a total sequence length of $2N$. The DFT will then evaluate $Q(\Omega)$ at $2N$ points, doubling the resolution. There is a practical limit to how many zeros you can add in order to increase the resolution. Oftentimes, increased resolution is only necessary within a small frequency interval. In this case a simple and straightforward method is to evaluate equation 1 directly over the frequency range you are interested in. In what follows, we give an efficient algorithm for doing this.

Using Euler's identity, we can split equation 1 into an equation for the real part, and an equation for the imaginary part.

$$\Re e\, Q(\Omega) = q[0] + q[1]cos(\Omega) + \sum_{n=2}^{N-1} q[n]cos(n\Omega) \tag{2}$$

$$\Im m\, Q(\Omega) = -q[1]sin(\Omega) - \sum_{n=2}^{N-1} q[n]sin(n\Omega) \tag{3}$$

We efficiently calculate $sin(n\Omega)$ and $cos(n\Omega)$ recursively, starting only with $sin\Omega$ and $cos\Omega$ by use of the Chebyshev polynomials. [2]

$$cos(n\Omega) = T_n(cos\Omega) \tag{4}$$

$$sin(n\Omega) = sin\Omega\, U_{n-1}(cos\Omega) \tag{5}$$

Knowing that for equations 4 and 5, $T_0 = 1$, $T_1 = cos\Omega$, $U_0 = 1$, and $U_1 = 2cos\Omega$, we can calculate all other T_n's and U_n's with the following formulas.

$$T_{n+1}(cos\Omega) = 2(cos\Omega)T_n(cos\Omega) - T_{n-1}(cos\Omega) \tag{6}$$

$$U_{n+1}(cos\Omega) = 2(cos\Omega)U_n(cos\Omega) - U_{n-1}(cos\Omega) \tag{7}$$

[2] The Chebyshev polynomials are a set of orthogonal polynomials that have found many uses in numerical analysis. A good reference is (sec. 13.3 and 13.4):

G. Arfken, Mathematical Methods for Physicists, 3rd ed., Academic Press, 1985.

Now we can calculate a subspectrum over a very small frequency range at an arbitrarily high resolution (much higher than the standard FFT), requiring only $cos\Omega$ and $sin\Omega$.

We have written a program, in C, for doing this, called **hrft** (see program listing). The program inputs are: the starting and ending frequency in Hz of the high resolution spectrum, the number of frequency points to calculate in the interval, the number of samples to read from the input file, the sampling rate (samples per second) for the data, and the names of the input and output files. The input file is assumed to be an ASCII text file with each data point on a separate line. Lines at the begining of the file that start with # are treated as comments and ignored. The output file is also an ASCII text file with each line containing the the real and imaginary parts of the FT point. The program is executed as follows:

hrft f1 f2 Nf N sps infile outfile
f1 = starting frequency [Hz]
f2 = ending frequency [Hz]
Nf = number of frequency points to calculate FT
N = number of samples to read from the input file
sps = samples per second
infile = input file
outfile = output file

Figure 1 shows part of the spectrum, produced by an FFT, from 2260 Hz to 2268 Hz, along with the spectrum produced by the high resolution Fourier transform (HRFT) over that frequency range.
Both methods used the same 2048 point data set. The FFT has a resolution of 4 Hz/bin, while the HRFT, calculated for 41 points, has a resolution of 0.2 Hz/bin. This is evident in the smoothness of the 41 point HRFT plot, as compared to the blocky 3 point FFT plot. The data set has a primary frequency component at 2265 Hz, so we can see that the HRFT method correctly identifies this frequency at its peak, while the peak of the FFT is off by 1 Hz. To get a resolution at least as good using the FFT, the data would have to be padded with zeros to a length of 65536 points.

In conclusion, if you are only interested in calculating the spectrum over a small frequency range, the HRFT may be more efficient than the FFT. The more narrow the frequency range the more attractive the HRFT becomes over the FFT. This is especially true when the computation has to be performed in an embedded system with limited memory resources.

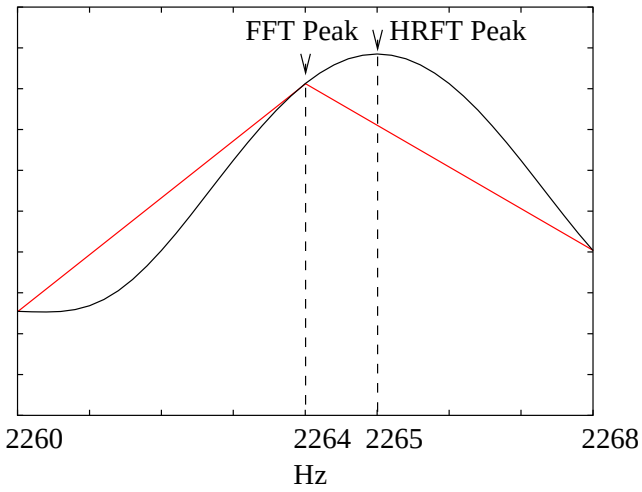

Figure 1: Comparison of FFT with HRFT using 2048 point data set having a major frequency component at 2265 Hz

---

Stefan and Richard design scientific instruments at Exstrom Laboratories in Longmont, Colorado. They both have a BS degreee in Electrical Engineering and an MS in Physics from the University of Louisville, Louisville, KY.

Code Listing of program hrft

```
/*
hrft
A C Program For Spectrum Magnification: When The FFT Is Not Enough
Version 1.0
March 27, 2003
by
Stefan Hollos, stefan@exstrom.com
Richard Hollos, richard@exstrom.com
Exstrom Laboratories LLC
P.O. Box 7651
Longmont, CO 80501
```

```
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#define MAXLINESIZE 128

/*
Calculates the Fourier transform over Nf points from frequency
f1 to frequency f2, of N (N>1) samples of a signal taken at
sps samples per second. The values of cos(n*Omega) & sin(n*Omega)
are calculated using Chebyshev polynomials.
*/

void hrft( int N, double *q, double Omega, double *Qr, double *Qi )
{
  int n;
  double C0, S0; /* cos(Omega) & sin(Omega) */
  double T0, T1, T2; /* Type I Chebyshev polynomials */
  double U0, U1, U2; /* Type II Chebyshev polynomials */
  double Cn, Sn; /* cos(n*Omega) & sin(n*Omega) */

  C0 = cos(Omega);
  S0 = sin(Omega);
  T0 = 1.0;
  T1 = C0;
  U0 = 1.0;
  U1 = 2.0*C0;

  /* this takes care of n = 0,1 */
  *Qr = q[0] + q[1]*T1;
  *Qi = q[1]*S0*U0;

  C0 *= 2.0;
  for( n = 2; n < N; ++n )
  {
```

```
    T2 = C0*T1 - T0;
    T0 = T1;
    T1 = T2;
    U2 = C0*U1 - U0;
    U0 = U1;
    U1 = U2;
    Cn = T2;
    Sn = S0*U0;
    *Qr += q[n]*Cn;
    *Qi += q[n]*Sn;
  } /* end for n */
} /* end function hrft */

/*****************************************************************/

int main( int argc, char *argv[] )
{
  double f1, f2; /* starting frequency f1, and ending frequency f2 [Hz] */
  int Nf; /* number of frequency points at which the FT is calculated */
  double df; /* frequency step size */
  int N; /* number of samples read from the input file, N>1 */
  int i, n; /* index */
  double sps; /* samples per second */
  double Omega; /* Omega = 2*Pi*f*T */
  double dOmega; /* Omega step size */
  FILE *fp; /* file pointer */
  char linebuff[MAXLINESIZE];
  double *q; /* array where samples are stored */
  double Qr, Qi; /* real & imaginary parts of FT */

  if( argc != 8 )
  {
    printf( "\nhrft calculates the Fourier transform at Nf equally\n");
    printf( "spaced points in the frequency interval [f1,f2]\n");
    printf( "usage: hrft f1 f2 Nf N sps infile outfile\n");
    printf( "  f1 = starting frequency [Hz]\n");
    printf( "  f2 = ending frequency [Hz]\n");
    printf( "  Nf = number of frequency  points to calculate FT\n");
```

```
  printf( "  N = number of samples to read from the input file\n");
  printf( "  sps = samples per second\n");
  printf( "  infile = input file\n");
  printf( "  outfile = output file\n");
  exit( -1 );
}

f1 = atof( argv[1] );
f2 = atof( argv[2] );
Nf = atoi( argv[3] );
N = atoi( argv[4] );
sps = atof( argv[5] );

if( N < 2 )
{
  printf( "N must be greater than 1\n" );
  exit( -1 );
}

/* open file to read */
if( ( fp = fopen( argv[6], "r" ) ) == NULL )
{
  perror( "Error opening input file" );
  exit( -1 );
}
printf( "Reading data file: %s\n\n", argv[6] );

/* Any line at the beginning of a data file that */
/* begins with a "#" is skipped */
for( n = 0; n < N; ++n )
{
  fgets( linebuff, MAXLINESIZE, fp );
  if( linebuff[0] != '#' ) break;
  printf( "%s", linebuff );
}

/* allocate a double array of size N */
q = (double *)malloc( N * sizeof(double) );
```

```
/* get the first number that's already in linebuff */
sscanf( linebuff, "%lf", &(q[0]) );
/* read the rest of the sample data */
for( n = 1; n < N; ++n )
{
  fscanf( fp, "%lf", &(q[n]) );
}
fclose( fp ); /* close input file */

if( ( fp = fopen( argv[7], "w" ) ) == NULL )
{
  perror("Error opening output file" );
  exit( -1 );
}
printf( "Opening output file: %s\n", argv[7] );

df = (f2-f1)/((double)(Nf-1));

/* write the header of the output file */
fprintf( fp, "# This file produced by program hrft\n" );
fprintf( fp, "# %20f  :f1, starting frequency [Hz]\n", f1 );
fprintf( fp, "# %20f  :f2, ending frequency [Hz]\n", f2 );
fprintf( fp, "# %20f  :df, frequency step size [Hz]\n", df );
fprintf( fp, "# %20d  :Nf, number of frequency points at which FT was calcula
fprintf( fp, "# %20d  :N, number of samples read from the input file, N>1\n",
fprintf( fp, "# %20f  :sps, samples per second\n", sps );
fprintf( fp, "# %s  :input file name\n", argv[6] );
fprintf( fp, "# Left column: real part of FT, Right column: imaginary part\n"

Omega = 2.0 * M_PI * f1 / sps;
dOmega = 2.0 * M_PI * df / sps;

for( i = 0; i < Nf; ++i )
{
  hrft( N, q, Omega, &Qr, &Qi );
  fprintf( fp, "%lf %lf\n", Qr, Qi );
  Omega += dOmega;
```

```
  } /* end for i */

  fclose( fp );
  free( q );
  return( 0 );
}
```