Maxima by Example:
Ch.11: Fast Fourier Transform Tools *

Edwin L. Woollett

August 13, 2009

Contents

*This is a live document. This version uses **Maxima 5.19.0**. Check `http://www.csulb.edu/~woollett/` for the latest version of these notes. Send comments and suggestions for improvements to `woollett@charter.net`

Preface

```
COPYING AND DISTRIBUTION POLICY
This document is part of a series of notes titled
"Maxima by Example" and is made available
via the author's webpage http://www.csulb.edu/~woollett/
to aid new users of the Maxima computer algebra system.

NON-PROFIT PRINTING AND DISTRIBUTION IS PERMITTED.

You may make copies of this document and distribute them
to others as long as you charge no more than the costs of printing.

These notes (with some modifications) will be published in book form
eventually via Lulu.com in an arrangement which will continue
to allow unlimited free download of the pdf files as well as the option
of ordering a low cost paperbound version of these notes.
```

Feedback from readers is the best way for this series of notes to become more helpful to new users of Maxima. *All* comments and suggestions for improvements will be appreciated and carefully considered.

```
LOADING FILES
The defaults allow you to use the brief version load(fft) to load in the
Maxima file fft.lisp.
To load in your own file, such as qfft.mac (used in this chapter),
using the brief version load(qfft), you either need to place
qfft.mac in one of the folders Maxima searches by default, or
else put a line like:

file_search_maxima : append(["c:/work3/###.{mac,mc}"],file_search_maxima )$

in your personal startup file maxima-init.mac (see Ch. 1, Introduction
to Maxima for more information about this).

Otherwise you need to provide a complete path in double quotes,
as in load("c:/work3/qfft.mac"),

We always use the brief load version in our examples, which are generated
using the Xmaxima graphics interface on a Windows XP computer, and copied
into a fancy verbatim environment in a latex file which uses the fancyvrb
and color packages.
```

```
  Maxima, a Computer Algebra System.
  Version 5.19.0 (2009) using Lisp GNU Common Lisp (GCL) GCL 2.6.8
  (aka GCL).  http://maxima.sourceforge.net/
```

The homemade function **fll(x)** (first, last, length) is used to return the first and last elements of lists (as well as the length), and is defined in the Ch.1 utility file **mbe1util.mac**. We will include a reference to this definition when working with lists.

This function is defined by:

```
fll(x) := [first(x),last(x),length(x)]$
declare(fll,evfun)$
```

The author would like to thank the Maxima developers for their friendly help via the Maxima mailing list.

11.1 Examples of the Use of the Fast Fourier Transform Functions fft and inverse_fft

We discuss the use of Maxima's fast Fourier transform package **fft.lisp**, which has been rewritten (effective with version 5.19) to allow list input and output (rather than being restricted to array input and output).

We first present five simple examples of using Maxima's fast Fourier transform functions **fft** and **inverse_fft**. We then discuss our notation and the utility functions available in the Ch.11 file **qfft.mac**. We then present a derivation of the discrete Fourier transformation pairs, using Maxima's conventions.

11.1.1 Example 1: FFT Spectrum of a Monochromatic Signal

We load in the Maxima package **fft.lisp** and also our own package **qfft.mac**.

```
(%i1) load(fft);
(%o1)  C:/PROGRA~1/MA89DF~1.0/share/maxima/5.19.0/share/numeric/fft.lisp
(%i2) load(qfft);
(%o2)                          c:/work3/qfft.mac
```

Our first example uses the simplest possible signal which still contains some information, a signal having one intrinsic frequency. We assume the signal is $\mathbf{F(t) = \cos(6\,\pi\,t)}$. This signal has an angular frequency $\boldsymbol{\omega = 6\,\pi = 2\,\pi\,f}$ where **f** is the frequency in Hertz (ie., in $\mathbf{sec^{-1}}$). Thus the frequency $\mathbf{f = 3\,sec^{-1}}$. We bind the symbol **e** to this signal expression for later use:

```
(%i3) e : cos ( 6*%pi*t )$
```

Let **ns** be the (even) number of function samples, and **fs** be the sampling frequency.

The **qfft** package function **nyquist(ns, fs)** computes the time interval between signal samples **dt = 1/fs**, the Nyquist integer **knyq = ns/2**, the frequency resolution **df = fs/ns**, and the Nyquist frequency **fnyq = knyq * df = fs/2**.

The product of the time interval **dt** between samples of the signal and the frequency resolution **df** is always the inverse of the total number of signal samples **ns**:

```
              dt * df = 1 / ns
```

We select **fs** and **ns** to satisfy two conditions. The only intrinsic frequency of the signal is $\mathbf{f0 = 3\,s^{-1}}$. **First** we need $\mathbf{fs > 2\,f0}$ which means that $\mathbf{fs > 6\,s^{-1}}$. **Secondly** we need the frequency resolution **df** to satisfy $\mathbf{df < f0}$, which will be satisfied if we choose **df** such that $\mathbf{f0 = 3\,df}$, or **df = 1 hertz**. But then **df = fs / ns = 1**, or **fs = ns**, so the first condition becomes $\mathbf{ns > 6}$, but we also need **ns** to be an even number of the form $\mathbf{2^m}$ for some integer **m**, so we choose **ns = 8**.

We then bind both **ns** and **fs** to the value **8** and assign **dt** (the sampling interval) to the first element of the list produced by the package function **nyquist(ns, fs)**, which also prints out **dt**, **knyq**, **fnyq**, and **df**.

```
(%i4) ( ns : 8, fs : 8 )$
(%i5) dt : first (nyquist (ns,fs));
sampling interval dt =  0.125
Nyquist integer knyq =  4
Nyquist freq fnyq =  4.0
freq resolution df =  1.0
(%o5)                                0.125
```

We then use the package function **sample(expr, var, ns, dvar)** to generate a list of floating point numbers as the expression **expr** is evaluated **ns** times, generating the list (with **dvar** being the time step **dt** here) **[F(0),F(dt),F(2*dt),...,F((ns-1)*dt)]** holding the values **F(m*dt)**, for **m = 0,1,2,...,ns-1**. More details of the syntax of the **qfft.mac** package functions can be found in Section 11.3. Recall that we have just bound **ns** and **dt** to numerical values and that the expression **e** depends on the variable **t**.

```
(%i6) flist : sample (e,t,ns,dt);
(%o6) [1.0, - 0.707, - 1.83691E-16, 0.707, - 1.0, 0.707, 5.51073E-16, - 0.707]
```

We see that elements three, **F(2*dt)**, and seven, **F(6*dt)**, are tiny numbers of the order of the default floating point errors, and are numerically equivalent to zero.

We first make a plot of **both** the given function (the signal) and a list of points **[m*dt, F(m*dt)]** constructed using the package function **vf (flist, dt)**. Recall that the signal is first sampled at **t = 0**.

```
(%i7) tflist : vf (flist,dt);
(%o7) [[0, 1.0], [0.125, - 0.707], [0.25, - 1.83691E-16], [0.375, 0.707],
          [0.5, - 1.0], [0.625, 0.707], [0.75, 5.51073E-16], [0.875, - 0.707]]
```

We let **tmax** be the "sampling time", the number of signal samples times the time interval between samples.

```
(%i8) tmax : ns*dt;
(%o8)                                  1.0
(%i9) plot2d([e ,[discrete,tflist]], [t,0,tmax],
             [style,[lines,3],[points,3,2,1]],
              [legend,false])$
```

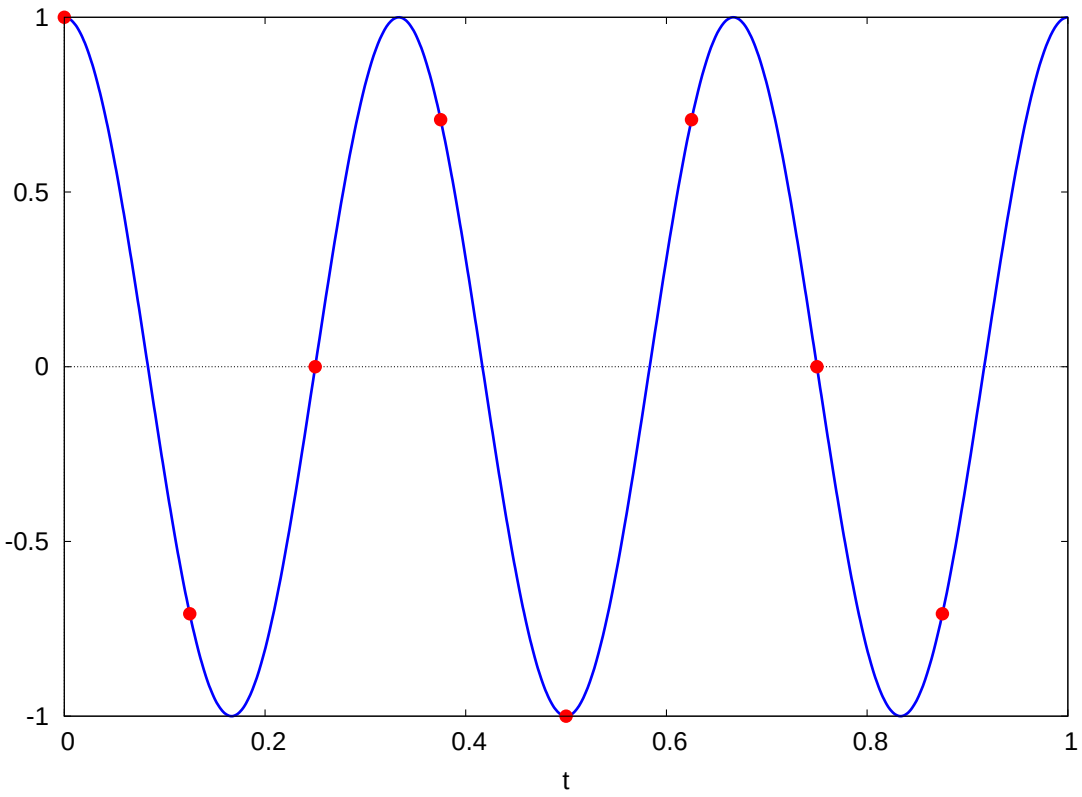

Figure 1: $\mathbf{F(t) = \cos(6\,\pi\,t)}$ with Sample Points

Now that we have looked at both the signal and sample signal values (on the same plot), we next look at the **fast Fourier frequency spectrum** implied by this single frequency signal sample. We first define **glist** to be the fast fourier transform of **flist**

```
(%i10) glist : fft (flist);
(%o10) [- 2.34662E-17, 1.11022E-16 - 3.30546E-17 %i,
1.52656E-16 %i - 4.59227E-17, 0.5 - 1.18171E-16 %i, 1.15312E-16,
2.16746E-16 %i + 0.5, - 1.52656E-16 %i - 4.59227E-17,
5.55112E-17 - 6.55197E-17 %i]
(%i11) fchop (%);
(%o11)              [0.0, 0.0, 0.0, 0.5, 0.0, 0.5, 0.0, 0.0]
```

in which we have used Maxima's fast fourier transform function **fft**. We then used our package function **fchop** to set tiny floating point numbers to zero. The **qfft** package function **current_small()** prints out the current setting of the **qfft.mac** parameter **_small%** used by **fchop**.

```
(%i12) current_small()$
     current default small chop value =  1.0E-13
```

The first element of **glist** corresponds to **k = 0**, and the list **glist** contains the values of the fast Fourier transform amplitudes **G(k*df)** for **k = 0, 1, 2, ..., ns -1**, where **df** is the frequency resolution (for this example, **df = 1 hertz**). We call the value **G(k*df)** the "fast Fourier transform amplitude" corresponding to the frequency **f = k*df**.

If we look at the chopped version, we see that the first non-zero fast fourier transform amplitude occurs at element four, which corresponds to **k = 3**, which corresponds to the frequency **f = k*df = 3 * 1 = 3 hertz**. In a later section exploring the basic ideas of the fast Fourier transform, we explain why all the usable spectrum information is contained in the interval **k = 0** to **k = knyq = 4**.

To make a simple point plot of the fourier amplitudes, we first use the package function **kg** to make a list of the points **[k, G(k*df)]** out to the Nyquist integer **knyq**. Since we want real numbers for a plot, this function takes the absolute value of each Fourier amplitude and also chops tiny numbers.

```
(%i13) kglist : kg (glist);
(%o13)        [[0, 0.0], [1, 0.0], [2, 0.0], [3, 0.5], [4, 0.0]]
(%i14) plot2d ( [discrete, kglist],[x,0,5],[y,0,0.8],
                  [style,[points,5,1,1]],[xlabel,"k"],
                  [ylabel," "],[gnuplot_preamble,"set grid;"])$
```

which produces the simple plot

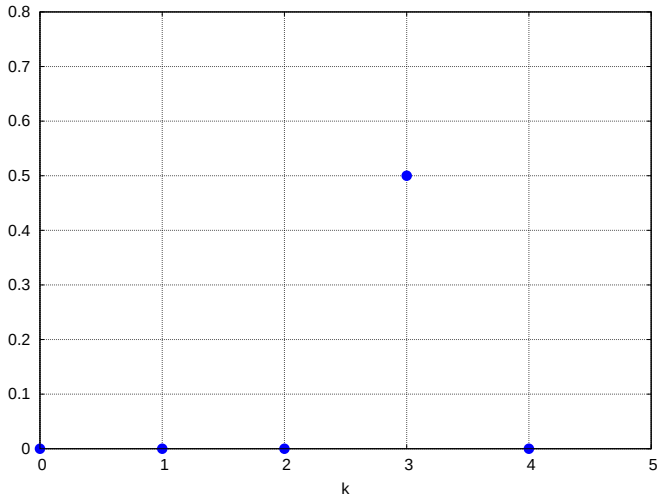

Figure 2: Spectrum of $\cos(6\,\pi\,t)$

We can also use **plot2d** to make a simple histogram, using **kglist**.

```
(%i15) vbars : makelist ( [discrete,
                  [[kglist[i][1],0],[kglist[i][1],kglist[i][2]]] ,
                      i,1,length(kglist) );
(%o15) [[discrete, [[0, 0], [0, 0.0]]], [discrete, [[1, 0], [1, 0.0]]],
[discrete, [[2, 0], [2, 0.0]]], [discrete, [[3, 0], [3, 0.5]]],
[discrete, [[4, 0], [4, 0.0]]]]
(%i16) plot2d ( vbars,[y,0,0.8],[style,[lines,5,1]],
               [ylabel," "],[xlabel," k "],[legend,false],
                [gnuplot_preamble,"set grid;"] )$
```

which produces the plot

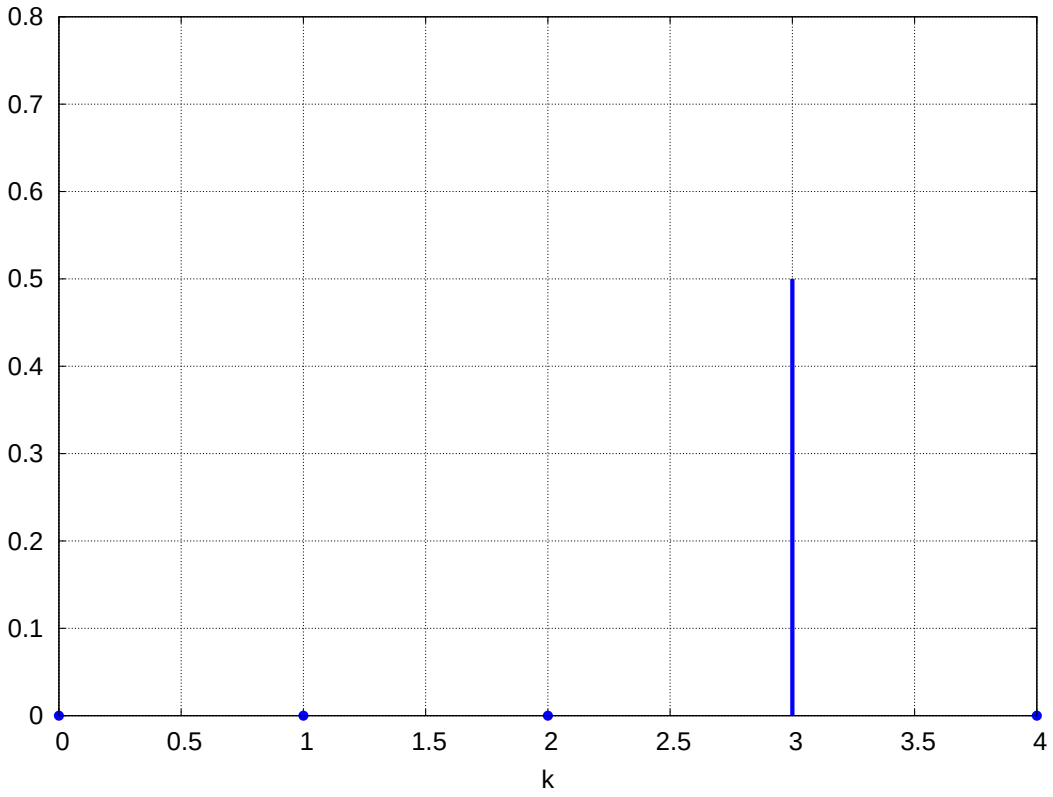

Figure 3: Line Spectrum of $\cos(6\,\pi\,t)$

The production of such a frequency space spectrum plot can be automated by accepting the fast Fourier transform list **glist** and defining a Maxima function **spectrum (glist, nlw, ymax)** which would be used with the specific syntax **spectrum (glist, 5, 0.8)** to obtain a histogram similar to the above but cleaner. (**nlw** is the line width and **ymax** is the vertical extent of the canvas starting at **`y = 0`**.)

We have designed **spectrum** to also allow the expanded syntax **spectrum (glist, nlw, ymax, k1, k2)** which zooms in on the interval **`k1 <= k <= k2`**, where **`k2 <= knyq`**. In addition, we have avoided, in **spectrum**, creating vertical bars when a fast Fourier transform amplitude is less than a very small number.

The entry

```
(%i17) spectrum (glist, 5, 0.8 )$
```

produces the plot

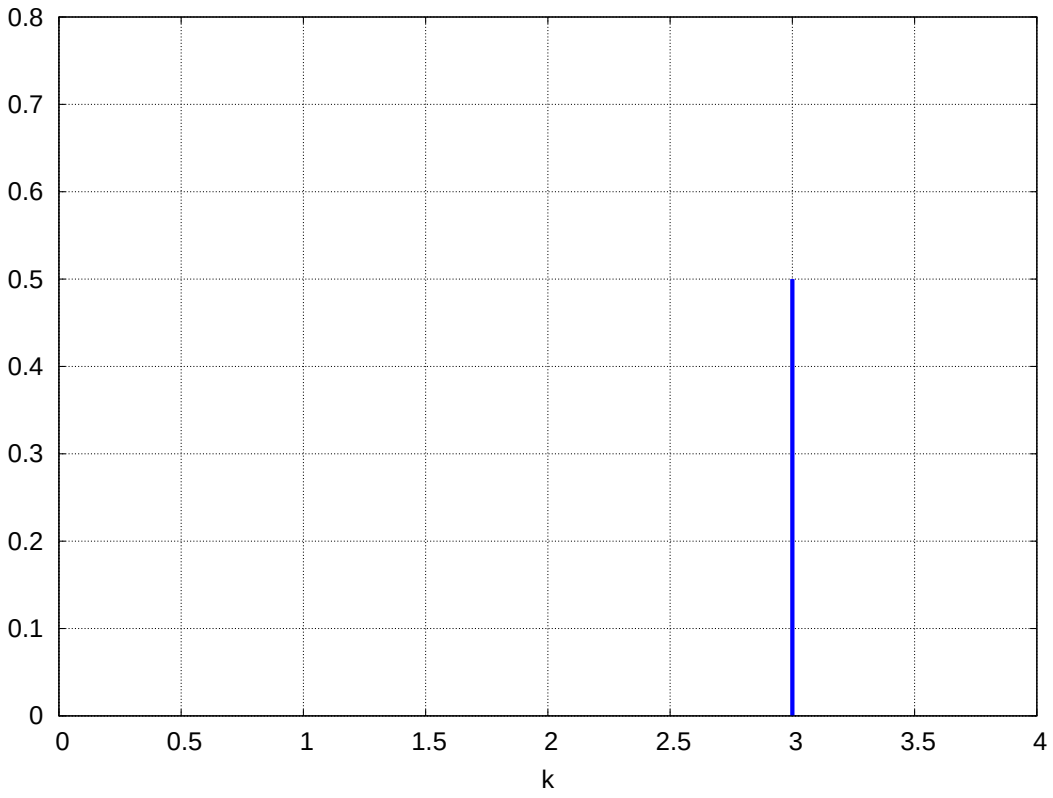

Figure 4: spectrum (glist, 5, 0.8)

We will use **spectrum**, included in **qfft.mac**, in our following examples as a quick way to look at the frequency spectrum.

Finally, let's use the Maxima function **inverse_fft** to apply the inverse fast Fourier transform to the list of fast Fourier transform amplitudes **glist**, and see how closely the result matches the original signal sample list **flist**.

```
(%i18) flist1 : inverse_fft(glist);
(%o18) [1.0, 1.11022E-16 %i - 0.707, 2.24949E-32 %i - 1.83691E-16,
6.77259E-17 %i + 0.707, - 1.0, 0.707 - 1.11022E-16 %i,
5.51073E-16 - 2.24949E-32 %i, - 6.77259E-17 %i - 0.707]
(%i19) fchop(%);
(%o19)      [1.0, - 0.707, 0.0, 0.707, - 1.0, 0.707, 0.0, - 0.707]
(%i20) fchop( flist);
(%o20)      [1.0, - 0.707, 0.0, 0.707, - 1.0, 0.707, 0.0, - 0.707]
(%i21) lmax ( abs (flist1 - flist));
(%o21)                           1.30049E-16
```

We see that **inverse_fft (glist)** recovers our original signal sample list **flist** to within floating point errors.

11.1.2 Example 2: FFT Spectrum of a Sum of Two Monochromatic Signals

Almost all signals will contain more than one intrinsic frequency, and to recover a portion of the frequency spectrum with fidelity, we need to satisfy the two conditions

$$\mathbf{fs} > \mathbf{2\,f_{high}} \quad \text{and} \quad \mathbf{df} \leq \mathbf{f_{low}}, \tag{11.1}$$

in which $\mathbf{f_{low}}$ is the lowest intrinsic frequency to be identified, and $\mathbf{f_{high}}$ is the highest intrinsic frequency to be identified. (Again, **fs** is the sampling frequency, and **df** is the frequency resolution).

We assume now that the signal is $\mathbf{F(t) = \cos(2\,\pi\,t) + \sin(4\,\pi\,t)}$. We thus have a signal with the frequencies $\mathbf{f_1 = 1\,s^{-1}}$ and $\mathbf{f_2 = 2\,s^{-1}}$. With $\mathbf{fs}$ being the sampling frequency, and $\mathbf{ns}$ being the number of signal samples, we require $\mathbf{fs > 2\,f_{high} = 4}$, as well as $\mathbf{df \leq f_{low} = 1}$. If we choose $\mathbf{f_{low} = 3\,df}$, then **`df = 1/3 = fs/ns`**, or **`fs = ns/3`**. So the first condtion then implies we need $\mathbf{ns/3 > 4}$, or $\mathbf{ns > 12}$. Since **`ns`** also should be equal to $\mathbf{2^m}$ for some integer $\mathbf{m}$, we choose **`ns = 16`**. Then **`fs = 16/3`** and **`df = fs/ns = 1/3 hertz`**.

```
(%i1) ( load(fft), load(qfft) )$
(%i2) e : cos(2*%pi*t) + sin(4*%pi*t)$
(%i3) (ns : 16, fs : 16/3)$
(%i4) dt : first (nyquist (ns,fs));
sampling interval dt =  0.188
Nyquist integer knyq =  8
Nyquist freq fnyq =  2.6667
freq resolution df =  0.333
(%o4)                                0.188
(%i5) flist : sample (e,t,ns,dt)$
(%i6) %,fll;
(%o6)                          [1.0, - 0.324, 16]
(%i7) tmax: ns*dt;
(%o7)                                3.0
(%i8) tflist : vf (flist,dt)$
(%i9) %,fll;
(%o9)                  [[0, 1.0], [2.8125, - 0.324], 16]
(%i10) plot2d([e ,[discrete,tflist]], [t,0,tmax],
             [style,[lines,3],[points,3,2,1]],
              [legend,false])$
```

We have used our utility function **`fll`** described in the preface, which returns the first and last element of a list, as well as the length of the list. The plot thus produced is

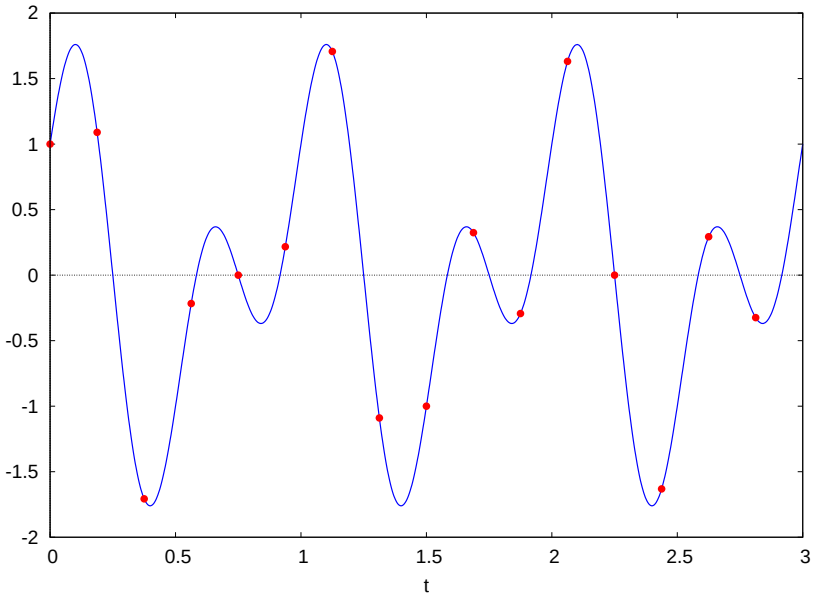

Figure 5: $\mathbf{\cos(2\,\pi\,t) + \sin(4\,\pi\,t)}$ with Sample Points

We next generate the list **glist** of fast Fourier transform amplitudes **G(k*df)** and use **spectrum** to look at the implied frequency spectrum.

```
(%i11) glist : fft (flist)$
(%i12) %,fll;
(%o12)          [- 7.94822E-17, 1.52656E-16 - 5.7151E-17 %i, 16]
(%i13) spectrum (glist,5,0.6)$
```

which produces the spectrum histogram:

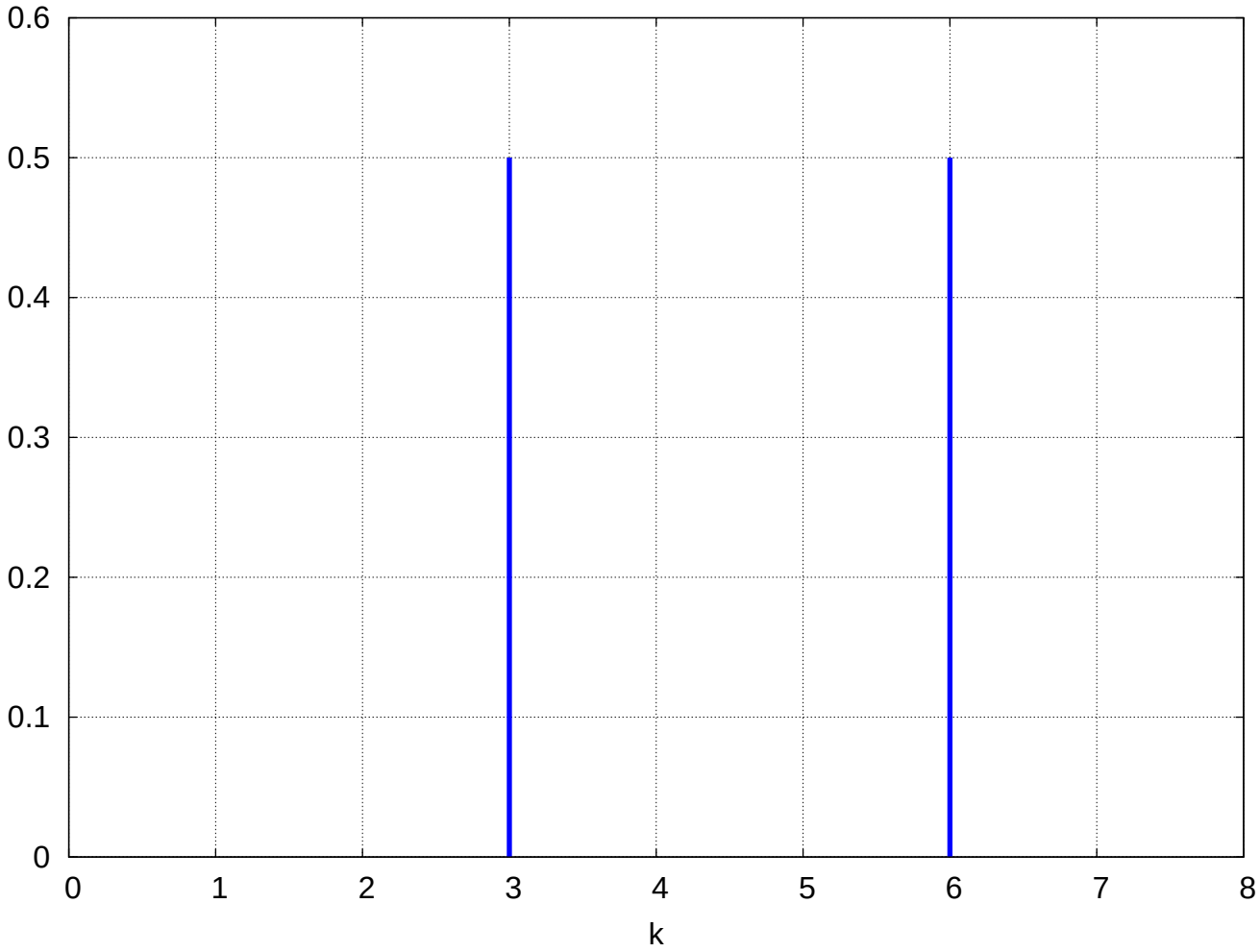

Figure 6: Spectrum of $\cos(2\,\pi\,t) + \sin(4\,\pi\,t)$

Since **df = 1/3**, the line at **k = 3** corresponds to the frequency **f = k*df = 3*(1/3) = 1 Hertz** and the line at **k = 6** corresponds to the frequency **f = k*df = 6*(1/3) = 2 Hertz**.

11.1.3 Example 3: FFT Spectrum of a Rectangular Wave

A rectangular wave as a function of time **t** with period equal to **64 sec** which cycles between plus and minus **1** can be constructed from **floor** and **mod** in the form: **rwave(t) := 2*mod(floor(t/32),2) - 1**.

For **t = 0, t/32 = 0, floor(t/32) = floor(0) = 0, mod(0,2) = 0**, so **rwave(0) = -1**.

```
(%i1) rwave(t):= 2*mod(floor(t/32),2) -1 $
(%i2) [floor(0),mod(0,2),rwave(0)];
(%o2)                              [0, 0, - 1]
```

For **t = 32, floor(32/32) = floor(1) = 1, mod(1,2) = 1, rwave(32) = 1**.

```
(%i3) [floor(1),mod(1,2),rwave(32)];
(%o3)                               [1, 1, 1]
```

For **t = 64, floor(64/32) = floor(2) = 2, mod(2,2) = 0, rwave(64) = -1**.

```
(%i4) [floor(2),mod(2,2),rwave(64)];
(%o4)                              [2, 0, - 1]
```

Hence **rwave(0 + 64) = rwave(0)** and **rwave(t)** has a period equal to **64 sec**.

We first look at the rectangular wave signal with a plot.

```
(%i5) plot2d(rwave(t),[t,0,128],[y,-1.5,1.5],
         [ylabel," "],[style,[lines,5]],
         [gnuplot_preamble,"set grid;set zeroaxis lw 2;"])$
```

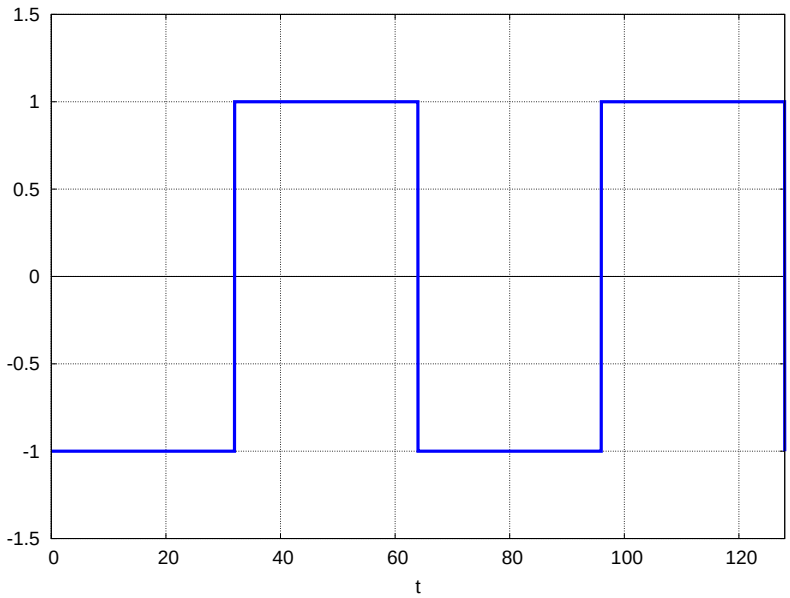

Figure 7: Rectangular Wave with Period 64 sec.

If we consider sampling this signal at intervals **dt = 1 sec**, the sampling frequency will be **fs = 1/dt = 1 hertz = 1 per sec = 64 per cycle**. The lowest intrinsic frequency of this signal is **f_low =** 1/64 hertz—, corresponding to the period of the rectangular wave. If we choose **4*df = f_low**, then **df = 1/256 = fs/ns = 1/ns**, so **ns = 256**. Due to the sharp corners of a rectangular wave we expect many high frequency components to be present in the spectrum.

We thus try the combination, **ns = 256**, and **fs = 1 hertz**.

```
(%i6) (ns:256, fs:1)$
(%i7) (load(fft),load(qfft) )$
(%i8) dt:first(nyquist(ns,fs));
sampling interval dt =  1.0
Nyquist integer knyq =  128
Nyquist freq fnyq =  0.5
freq resolution df =  0.00391
(%o8)                                1.0
(%i9) flist : sample(rwave(t),t,ns,dt)$
(%i10) fll (flist);
(%o10)                          [- 1.0, 1.0, 256]
(%i11) makelist (flist[i],i,1,10);
(%o11) [- 1.0, - 1.0, - 1.0, - 1.0, - 1.0, - 1.0, - 1.0, - 1.0, - 1.0, - 1.0]
(%i12) tmax: ns*dt;
(%o12)                               256.0
(%i13) tflist : vf (flist,dt)$
(%i14) fll (tflist);
(%o14)                  [[0, - 1.0], [255.0, 1.0], 256]
(%i15) plot2d([rwave(t) ,[discrete,tflist]], [t,0,tmax],
              [y,-1.5,1.5],[ylabel," "],
             [style,[lines,3],[points,1,0,1]],
              [legend,false])$
```

The **sample_plot** invocation produces the black points of the sample on top and bottom of the rectangular wave shown here:

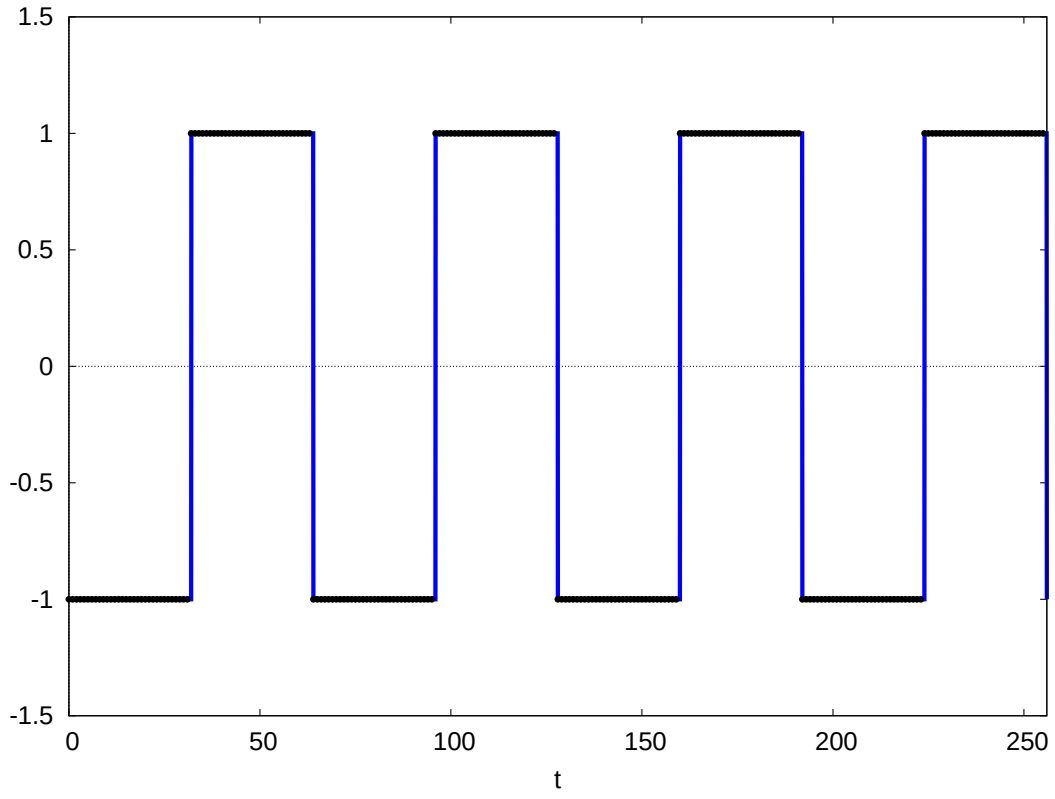

Figure 8: Black Sample Points with Rectangular Wave

Next we look at the signal frequency spectrum from **k = 0** to **k = knyq = 128**. Since **f = k*df**, and **df = 1/256 hertz**, the low fundamental intrinsic frequency **f_0 = 1/64 hertz** will be located at **k = 4**, and the maximum possible frequency component will be **fnyq = knyq*df = 0.5 hertz = 32*f_0**. The spectrum plot is generated by passing the list **flist** containing **ns = 256** signal samples, along with **nlw = 3**, and **ymax = 0.7** to **spectrum**:

```
(%i16) glist : fft (flist)$
(%i17) %,fll;
(%o17)                              [0.0, 0.0, 256]
(%i18) spectrum (glist,3,0.7)$
```

which produces the spectrum plot:

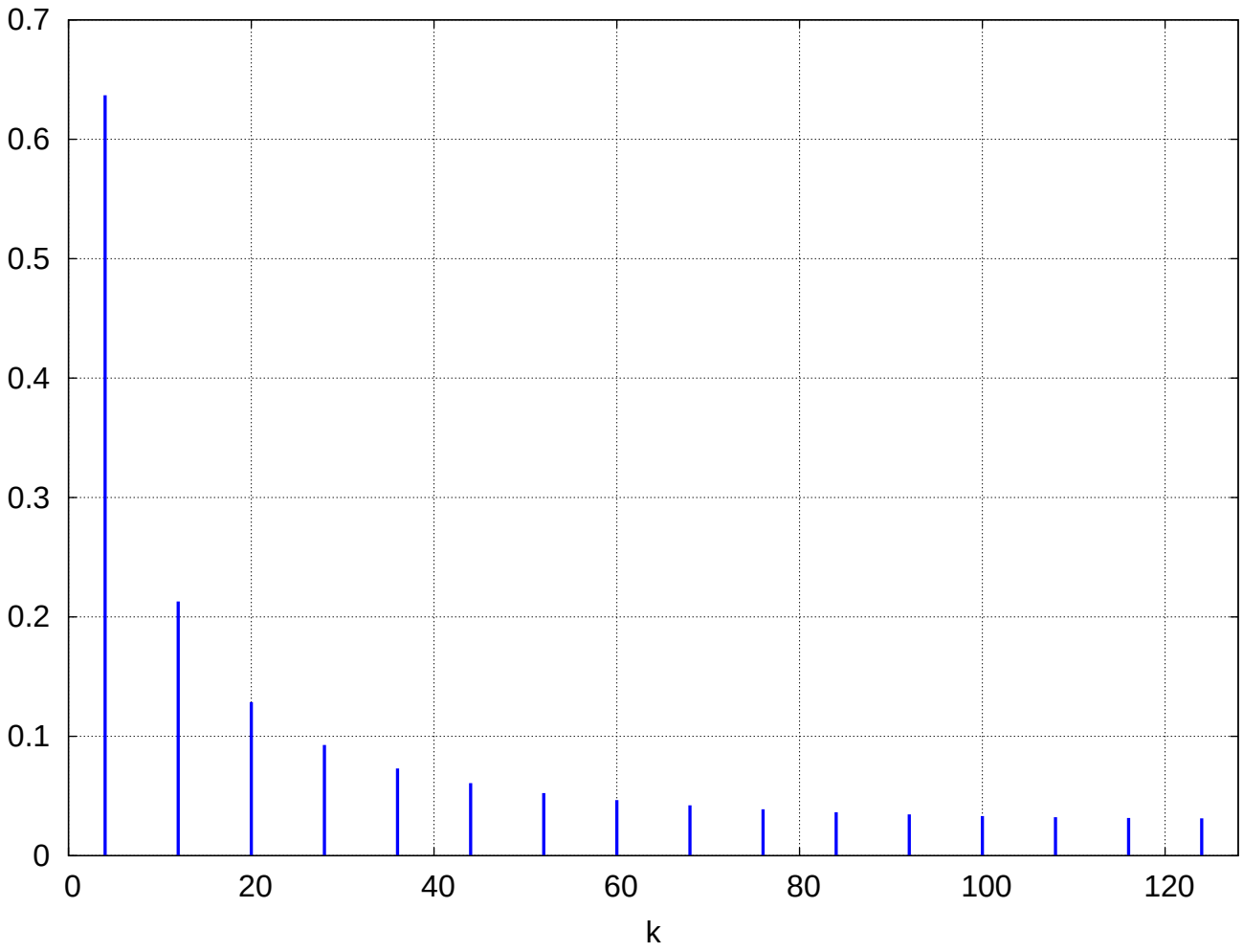

Figure 9: Spectrum from k = 0 to knyq = 128

The plot shows lines at **k = 4, 12, 20, 28, 36, ...** Since **f = k*df** and **f_0 = 4*df** is the fundamental frequency, the frequencies present are **f_0, 3*f_0, 5*f_0, 7*f_0,...** which is the "fundamental" plus odd harmonics of the fundamental. We know there must be high frequency components to account for the sharp corners of the signal.

11.1.4 Example 4: FFT Spectrum Sidebands of a Tone Burst Before and After Filtering

A tone burst signal consisting of a sine wave begins at **t = 0** and ends abruptly after forty five seconds. The fast Fourier frequency spectrum of such a signal, when sampled over the duration of the sine wave plus some time following the sine wave end, will contain "sideband" frequencies above and below the intrinsic frequency of the sine wave. Many signal filters can be used to allow the easy identification of the intrinsic sine wave frequency by surpressing the sideband frequencies. We use for the tone burst the sine wave: $\sin(2\,\pi\,t/5)$ during the time interval $0 \leq t \leq 45\,\mathrm{sec}$, and then abruptly dropping to **0** for $t > 45\,\mathrm{sec}$. The intrinsic frequency of the sine wave is **f0 = 1/5 hertz**, and the corresponding period is **5 sec**, thus the sine wave lasts for nine periods. We will use the following idealized model of such a signal by ignoring the actual time needed to end the sine wave.

```
(%i1) sig(t) := if t < 45 then sin(2*%pi*t/5.0)  else 0$
```

We will use this definition only for $t \geq 0$ in our work. With **fs** the sampling frequency, if we want about **10 samples/cycle** (ie., **10 per 5 sec**), then we want **period*fs = 10**, or **fs** roughly equal to **2 hertz**. We want the intrinsic, frequency **f0** (**0.2 hertz**) to not be too close to the left end of the frequency spectrum plot so we can see the low frequency sidebands as well as the higher frequency sidebands.

Suppose we try requiring that **f0 = 0.2 hertz = 50*df = 50*fs/ns**. Solving for the number of signal samples **ns**, we get **ns = 50 * 2 * 5 = 500**. To get a power of **2** we choose the closest such quantity, **ns = 512**. We then return to our **f0** location requirement to solve for the resulting value of **fs** to get **fs = 512/250 = 2.048 hertz**.

```
(%i2) load (qfft)$
(%i3) (ns:512,fs:2.048)$
(%i4) dt : first(nyquist(ns,fs));
sampling interval dt =  0.488
Nyquist integer knyq =  256
Nyquist freq fnyq =  1.024
freq resolution df =  0.004
(%o4)                                0.488
(%i5) tmax : ns*dt;
(%o5)                                250.0
```

We can plot a Maxima function defined with the **if then else** construct without problem since **plot2d** evaluates its arguments.

```
(%i6) plot2d ( sig(t),[t,0,tmax],[y,-1.5,1.5],
         [style,[lines,1,0]],[ylabel," "],[nticks,100],
          [legend,false],[gnuplot_preamble,"set grid;"])$
```

which produces the plot:

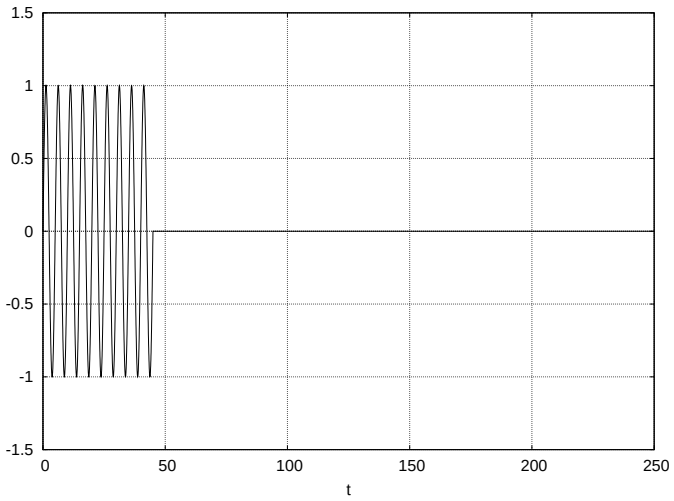

Figure 10: Sine Wave Tone Burst, Period = 5 sec, Duration = Nine Periods

Using the **qfft** package function **sample** with a Maxima function defined with the **if...then** construct produces a list each element of which includes **if then else**, which can be used with **plot2d**. However, we will include an extra evaluation in defining our **flist** so we can see the numerical values as a list. Note that the syntax **expr, fll;** causes an extra evaluation, so to be careful we need to use the syntax **fll (expr)**.

```
(%i7) flist : sample (sig(t),t,ns,dt)$
(%i8) fll (flist);
(%o8) [if 0.0 < 45.0 then 0.0 else 0.0,
                                  if 249.51 < 45.0 then - 0.576 else 0.0, 512]
(%i9) flist : ev (flist)$
(%i10) fll (flist);
(%o10)                          [0.0, 0.0, 512]
(%i11) makelist(flist[i],i,1,10);
(%o11) [0.0, 0.576, 0.942, 0.964, 0.634, 0.0736, - 0.514, - 0.914, - 0.981,
                                                                    - 0.69]
(%i12) tflist : vf (flist,dt)$
(%i13) fll (tflist);
(%o13)                  [[0, 0.0], [249.51, 0.0], 512]
```

We now use **plot2d** to show our sample points on top of our tone burst signal (both in black).

```
(%i14) plot2d([sig(t) ,[discrete,tflist]], [t,0,tmax],
              [y,-1.5,1.5],[ylabel," "],
             [style,[lines,1,0],[points,1,0,1]],
              [legend,false])$
```

which produces the plot:

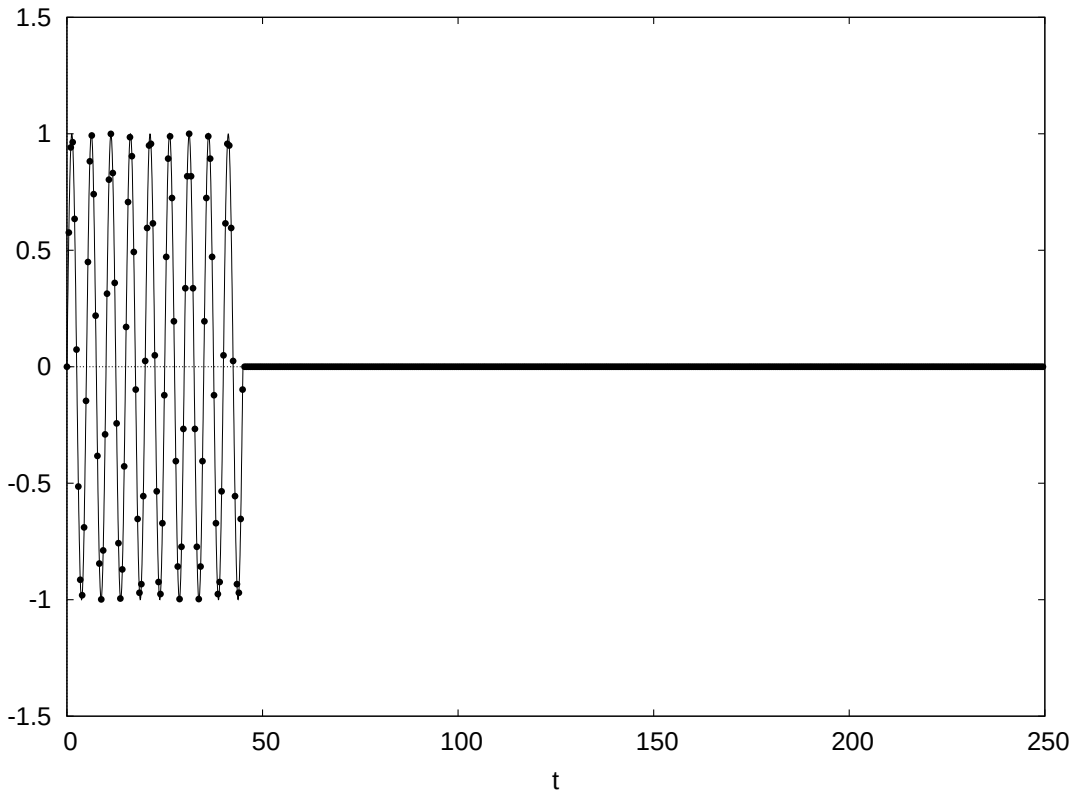

Figure 11: Signal and Sample Points Which Include 0's for $\mathbf{t > 45\,sec}$

We now plot the unfiltered tone burst spectrum after generating **glist** with **fft**.

```
(%i15) load (fft)$
(%i16) glist : fft (flist)$
(%i17) fll (glist);
(%o17)              [- 8.08763E-5, 0.00174 - 0.00286 %i, 512]
(%i18) spectrum (glist, 2, 0.1 )$
```

which produces the spectrum plot: We see strong sideband frequencies on both the low and high side of the intrinsic

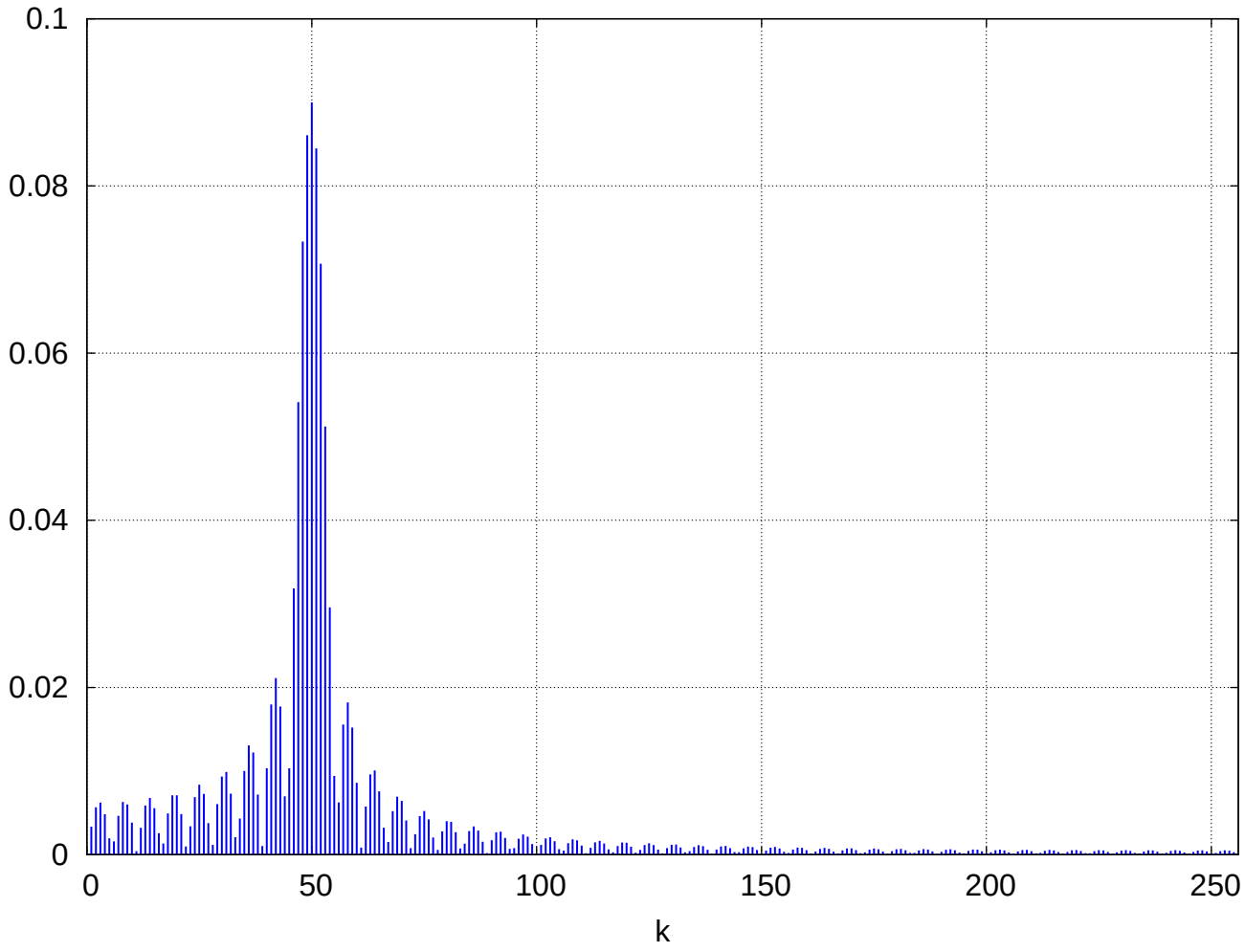

Figure 12: Frequency Spectrum of Unfiltered Tone Burst

frequency **f0 = 50*df**, which is the highest peak at **k = 50** (**f = k*df = 50*(1/250) = 1/5 = 0.02**). The sidebands are a mathematical artifact of the finite duration of the sine wave tone burst with its abrupt beginning and end.

A windowing filter which smooths out the beginning and end of the "burst envelope" is used by defining a smoothed envelope signal as the product of the unsmoothed signal and a suitable windowing function. The von Hann window employs a $\mathbf{sin^2}$ pinch envelope.

```
(%i19) hannw(x,m) := sin(%pi*(x-1)/(m-1))^2$
(%i20) sig_w(t) := hannw(t,45)*sig(t)$
(%i21) plot2d ( sig_w(t),[t,0,tmax],[y,-1.5,1.5],
         [style,[lines,1,0]],[ylabel," "],[nticks,100],
          [legend,false],[gnuplot_preamble,"set grid;"])$
```

which shows the filtered tone burst

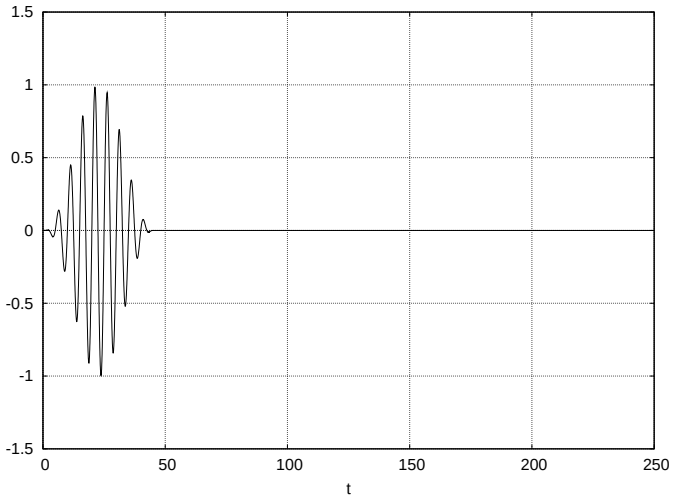

Figure 13: Filtered Tone Burst

Using now the filtered tone burst signal **`sig_w(t)`**, we construct a filtered signal sample list **`flist_w`** and look at the altered fast Fourier spectrum (remember **`dt`** has been bound to **`0.488`** and **`ns`** to **`512`**):

```
(%i22) flist_w : sample (sig_w(t),t,ns,dt)$
(%i23) fll (flist_w);
(%o23) [0.00509 (if 0.0 < 45.0 then 0.0 else 0.0),
                          0.799 (if 249.51 < 45.0 then - 0.576 else 0.0), 512]
(%i24) flist_w : ev (flist_w)$
(%i25) fll (flist_w);
(%o25)                              [0.0, 0.0, 512]
(%i26) makelist (flist_w[i],i,1,5);
(%o26)           [0.0, 7.68319E-4, 2.63668E-6, 0.00106, 0.00293]
(%i27) glist_w : fft (flist_w)$
(%i28) makelist (glist_w[i],i,1,5);
(%o28) [1.59068E-5, 3.93191E-6 - 1.99372E-5 %i, - 1.83151E-5 %i - 1.84837E-5,
                        2.09989E-6 %i - 2.59175E-5, 1.69797E-5 %i - 9.67992E-6]
(%i29) spectrum (glist_w, 2, 0.1)$
```

and we see deletion of most of the sideband frequency peaks:

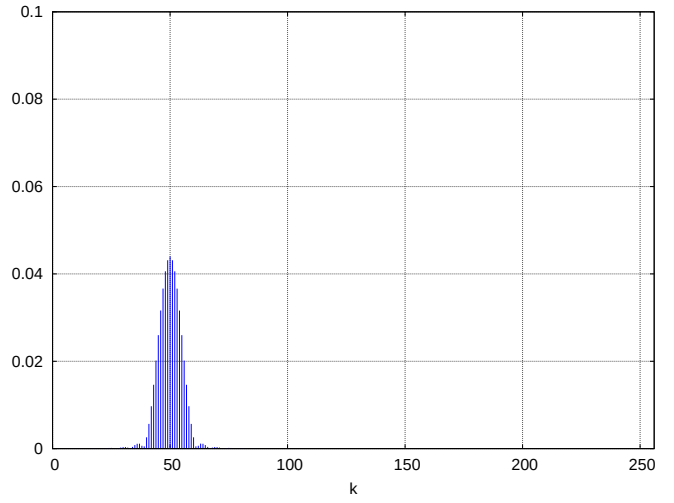

Figure 14: Frequency Spectrum of the Filtered Tone Burst

11.1.5 Example 5: Cleaning a Noisy Signal using FFT Methods

We use the same signal as used in Example 2, Sec. (11.1.2), but add some random noise to it. Without the noise, the signal is $\mathbf{F(t) = \cos(2\,\pi\,t) + \sin(4\,\pi\,t)}$. Thus the clean signal contains the frequencies $\mathbf{f_1 = 1\,s^{-1}}$ and $\mathbf{f_2 = 2\,s^{-1}}$, and two corresponding periods $\boldsymbol{\tau_1 = 1/f_1 = 1\,sec}$ and $\boldsymbol{\tau_2 = 1/f_2 = 0.5\,sec}$. To get a noisy looking signal we have to sample and add noise a lot of times within one or two periods of the clean signal, where we use the maximum of the intrinsic periods. Let's try $\mathbf{N = 512}$ samples over a time **tmax = 2 sec**. Then $\mathbf{N\,\delta t = 2\,sec}$, so **fs = 1/dt = N/2 = 256**. The first condition on the sampling frequency is that $\mathbf{f_s > 2\,f_{high}}$, or $\mathbf{fs > 4\,s^{-1}}$, which is certainly satisfied. The second condition on **fs** is $\mathbf{\delta f < f_{low}}$, or $\mathbf{df < 1\,s^{-1}}$, or $\mathbf{fs/ns < 1\,s^{-1}}$, or $\mathbf{fs < ns}$, which is also true. Hence we try **ns = 512, fs = 256**.

```
(%i1) e : cos(2*%pi*t) + sin(4*%pi*t)$
(%i2) (load(fft), load(qfft))$
(%i3) [ns:512,fs:256]$
(%i4) dt : first(nyquist(ns,fs));
sampling interval dt =  0.00391
Nyquist integer knyq =  256
Nyquist freq fnyq =  128.0
freq resolution df =  0.5
(%o4)                                0.00391
(%i5) flist : sample(e,t,ns,dt)$
(%i6) %,fll;
(%o6)                          [1.0, 0.951, 512]
(%i7) flist_noise : makelist(flist[j]+0.3*(-1.0+random(2.0)),j,1,ns)$
(%i8) %,fll;
(%o8)                        [1.2483, 1.0073, 512]
(%i9) tflist_noise : vf (flist_noise,dt)$
(%i10) %,fll;
(%o10)               [[0, 1.2483], [1.9961, 1.0073], 512]
(%i11) plot2d ([discrete,tflist_noise],[y,-2,2],
                [style,[lines,1]],[ylabel," "])$
```

which produces the noisy signal plot

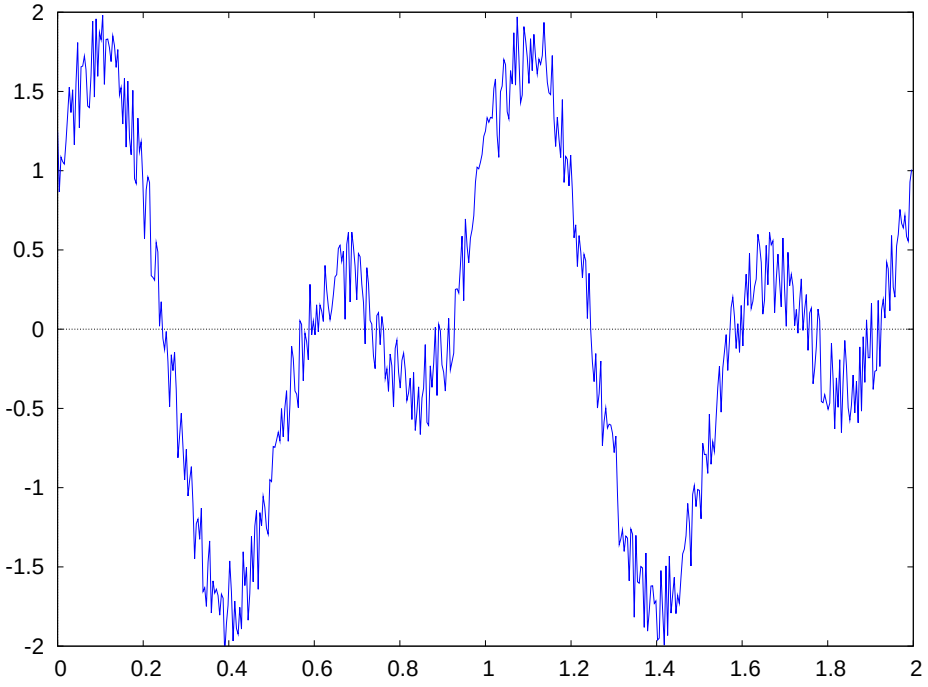

Figure 15: Noisy Signal

One way to “clean” this signal is to set small numbers in the fast Fourier transform to zero and inverse transform back to the time domain using **inverse_fft**.

In order to “chop” small numbers in the fast Fourier transform amplitude list, we need to produce that list, which we call **glist_noise** from the noisy signal sample **flist_noise** by using **fft**.

```
(%i12) glist_noise : fft (flist_noise)$
(%i13) %,fll;
(%o13)              [- 0.0022, 5.09918E-4 %i - 0.00338, 512]
```

Before “chopping” small numbers in **glist_noise**, we take a look at the fast Fourier frequency spectrum implied by **glist_noise**.

```
(%i14) spectrum (glist_noise,2,0.6)$
```

which produces the plot

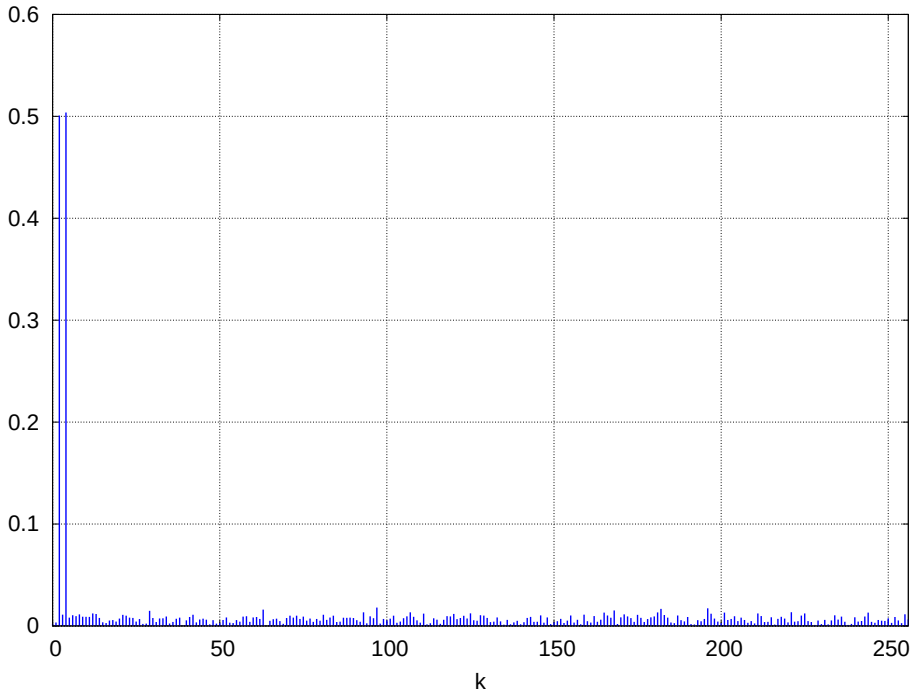

Figure 16: Noisy Signal Frequency Spectrum

The dominant lines are still those corresponding to the two intrinsic frequencies of the clean signal we started with, but there are many more frequencies present due to the noise.

To see more clearly the dominant line region, we show another plot with the integer **k** in the range **(0, 10)**:

```
(%i15) spectrum (glist_noise,4,0.6,0,10)$
```

which produces the plot:

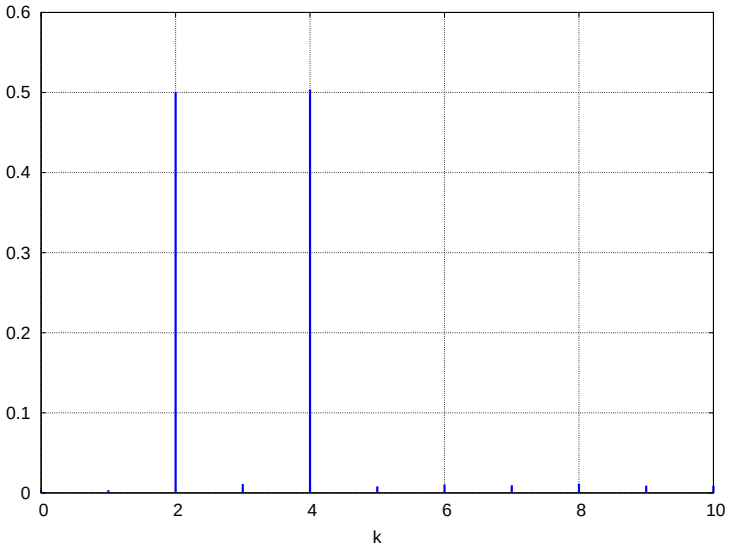

Figure 17: Noisy Signal Frequency Spectrum with k in (0,10)

We now use **fchop1** with the value **0.2** to set small floating point numbers less than **0.2** to zero in the frequency space list **glist_noise**, and again use **spectrum** to look at the frequency spectrum associated with the chopped frequency space list.

```
(%i16) glist_noise_chop : fchop1(glist_noise,0.2)$
(%i17) %,fll;
(%o17)                          [0.0, 0.0, 512]
(%i18) spectrum (glist_noise_chop,2,0.6)$
```

which produces the plot

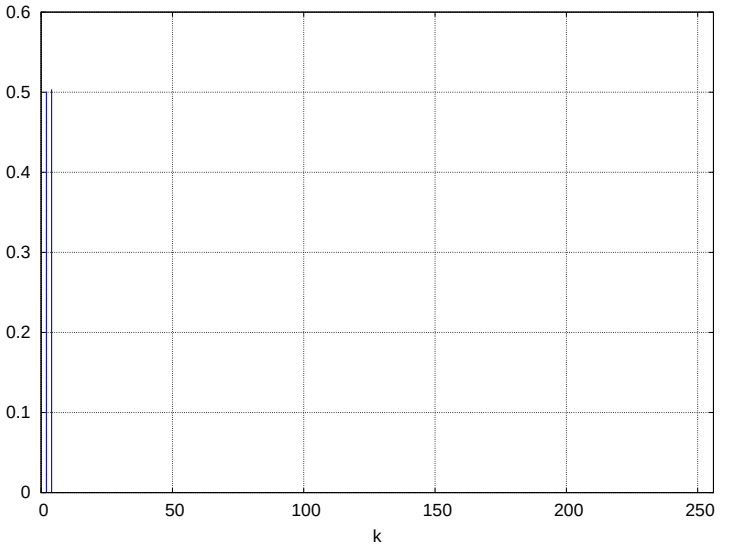

Figure 18: Chopped Frequency Spectrum

Here is the cleaned spectrum in the range **k = (0,10)**:

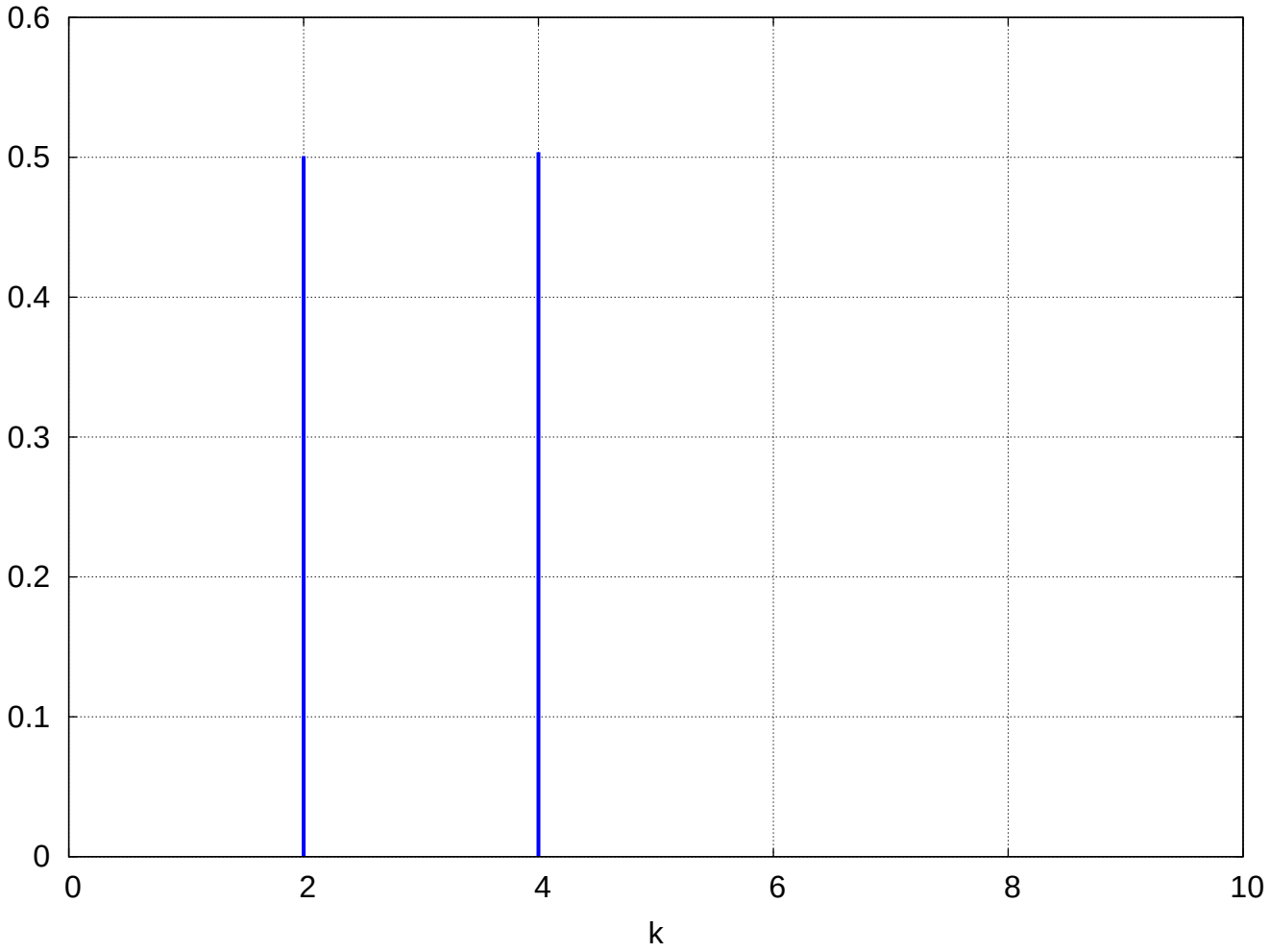

Figure 19: Chopped Frequency Spectrum for k in (0,10)

We now create a cleaned up signal list using **inverse_fft** on the chopped glist, transforming back to the time domain.

```
(%i19) flist_clean : inverse_fft ( glist_noise_chop )$
(%i20) %,fll;
(%o20)      [2.22045E-16 %i + 1.0016, 0.952 - 3.16992E-15 %i, 512]
(%i21) flist_clean : realpart(flist_clean)$
(%i22) %,fll;
(%o22)                        [1.0016, 0.952, 512]
```

Since the inverse Fourier transform will often include small imaginary parts due to floating point error, we took care to take the real part of the returned list before looking at the cleaned up signal. We now construct the list of points **[t, F_clean]**:

```
(%i23) tflist_clean : vf ( flist_clean, dt )$
(%i24) %,fll;
(%o24)                 [[0, 1.0016], [1.9961, 0.952], 512]
```

and plot first just the cleaned up signal points

```
(%i25) plot2d ([discrete,tflist_clean],[y,-2,2],
                [style,[lines,1]],[ylabel," "])$
```

which produces

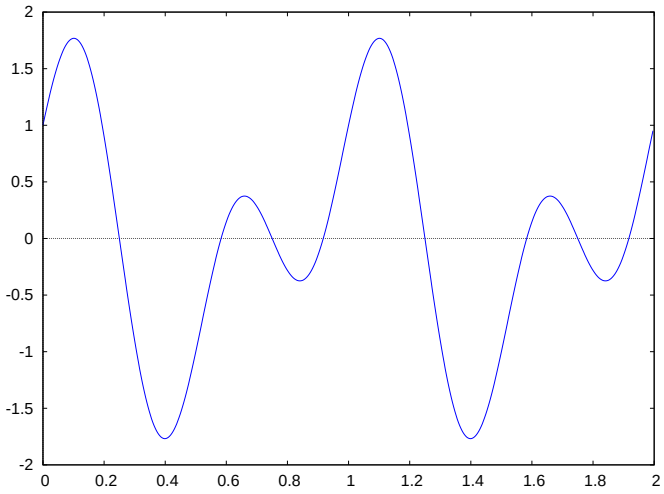

Figure 20: Cleaned Signal Points

We now show both the cleaned list points together with the original clean two frequency signal we started with, to show that the cleaned up points lie right on top of the original two frequency signal.

```
(%i26) plot2d([e ,[discrete,tflist_clean]], [t,0,2],
             [style,[lines,1],[points,1,0,1]],
              [legend,false])$
```

which produces the plot

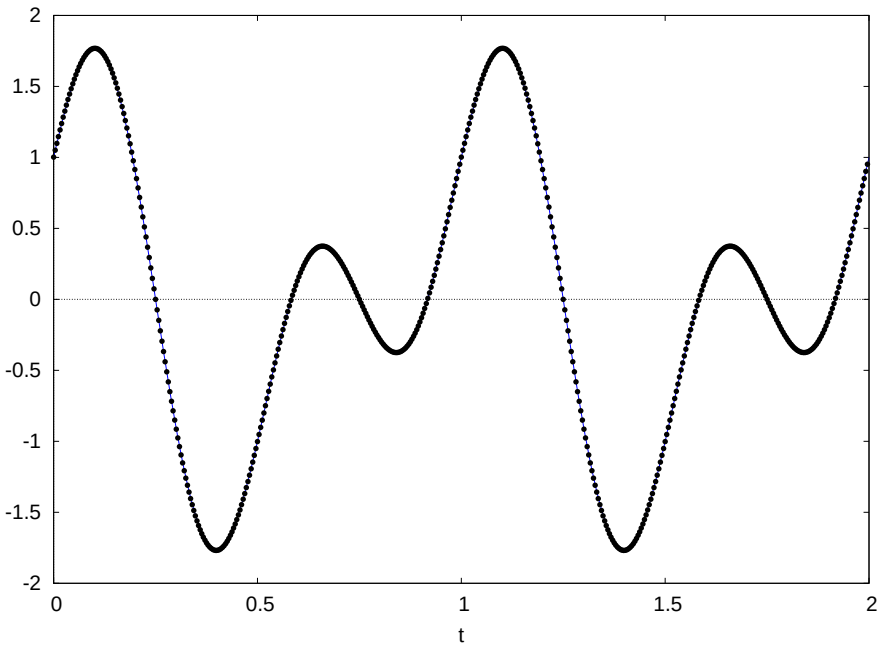

Figure 21: Cleaned Signal Points on Top of Original Clean Signal

We see that the inverse fast Fourier transform of the chopped glist frequency spectrum yields a cleaned up signal, as desired.

11.2 Our Notation for the Discrete Fourier Transform and its Inverse

Given a real valued signal $\mathbf{F(t)}$ which is to be sampled $\mathbf{N}$ times at the moments (separated by equal time intervals $\boldsymbol{\delta t}$) $\mathbf{t = 0}$, $\boldsymbol{\delta t}$, $\mathbf{2\,\delta t}$, ..., $\mathbf{(N-1)\,\delta t}$, one needs to select the two parameters $\mathbf{N = 2^m}$ and $\mathbf{f_s}$, where the latter is called the sampling frequency. The sampling time interval is then given by

$$\boldsymbol{\delta t} = \frac{\mathbf{1}}{\mathbf{f_s}} \tag{11.2}$$

(or else use the data determined value of $\boldsymbol{\delta t}$ to calculate $\mathbf{f_s}$). Note that the fast fourier transform algorithm used by Maxima assumes that the number of signal samples $\mathbf{N}$ is some integral power of 2, $\mathbf{N = 4, 8, 16, 32, 64}$, ... so an experimental sample might have to be padded with zeroes to achieve this condition on $\mathbf{N}$. The sampling frequency $\mathbf{f_s}$ should be greater than twice the highest frequency component to be identified in the signal and the frequency resolution $\boldsymbol{\delta f}$ should be smaller than the lowest frequency to be identified in the signal. Given the sampling frequency $\mathbf{f_s}$ and the number of signal samples $\mathbf{N}$, the frequency resolution $\boldsymbol{\delta f}$ is given by

$$\boldsymbol{\delta f} = \frac{\mathbf{f_s}}{\mathbf{N}} \tag{11.3}$$

We will motivate this definition below. Assuming this definition, we then require that

$$\frac{\mathbf{f_s}}{\mathbf{N}} < \mathbf{f_{low}} \tag{11.4}$$

The sampling frequency $\mathbf{f_s}$ thus needs to satisfy the two conditions:

$$\mathbf{2\,f_{high} < f_s < N\,f_{low}} \tag{11.5}$$

A convenient choice which automatically satisfies the low frequency part of these conditions is to arrange that

$$\mathbf{f_{low} = n\,\delta f}, \tag{11.6}$$

where $\mathbf{n = 3}$ or $\mathbf{4}$, say. Then that choice determines the frequency resolution to be used $\boldsymbol{\delta f} = \mathbf{f_{low}/n}$, and from the definition of $\boldsymbol{\delta f}$, Eq.(11.3), this requires that

$$\mathbf{f_s = N\,f_{low}/n} \tag{11.7}$$

and then Eq.(11.5) implies the condition on $\mathbf{N}$:

$$\mathbf{N > \frac{2\,n\,f_{high}}{f_{low}}} \tag{11.8}$$

In the simple case that $\mathbf{f_{low} = f_{high} = f_0}$ Eq.(11.7) becomes

$$\mathbf{f_s = N\,\frac{f_0}{n}}, \tag{11.9}$$

Eq.(11.5) becomes

$$\mathbf{N > 2\,n}, \tag{11.10}$$

and

$$\boldsymbol{\delta f} = \frac{\mathbf{f_0}}{\mathbf{n}} \tag{11.11}$$

In Example 1, the signal frequency is $\mathbf{f_0 = 3}$, and we chose **n = 3**. Then, Eq.(11.9) implies that $\mathbf{f_s = N}$, Eq.(11.10) implies that $\mathbf{N > 6}$, and Eq.(11.11) implies that $\boldsymbol{\delta f} = \mathbf{1}$. Since we need $\mathbf{N = 2^m}$ as well, we chose $\mathbf{f_s = N = 8}$.

In Example 2, $\mathbf{f_{low} = 1}$ and $\mathbf{f_{high} = 2}$, and we again chose **n = 3**. Then, Eq.(11.6) implies that $\boldsymbol{\delta f} = \mathbf{1/3}$, Eq.(11.7) implies that $\mathbf{f_s = N/3}$, and Eq.(11.8) implies that $\mathbf{N > 12}$. Since we need $\mathbf{N = 2^m}$ as well, we chose $\mathbf{N = 16}$ and this forces $\mathbf{f_s = 16/3}$.

The $\mathbf{N}$ real numbers $\mathbf{F(0)}$, $\mathbf{F(\delta t)}$, $\mathbf{F(2\,\delta t)}$, ..., $\mathbf{F(m\,\delta t)}$, ..., $\mathbf{F((N-1)\,\delta t)}$ can be used to define $\mathbf{N}$ complex numbers $\mathbf{G(k\,\delta f)}$, where $\mathbf{k = 0}$, $\mathbf{1}$, ..., $\mathbf{(N-1)}$, according to (this incorporates Maxima's conventions):

$$\mathbf{G(k\,\delta f) = \frac{1}{N} \sum_{m=0}^{N-1} F(m\,\delta t)\, e^{-2\,\pi\, i\, m\, k/N}} \tag{11.12}$$

The Maxima conventions include where to put the factor of $\mathbf{1/N}$ and what sign to use in the exponent argument. Our notation uses $\mathbf{i}$ to stand for the pure imaginary number $\sqrt{-1}$. Common engineering notation uses $\mathbf{j}$ for $\sqrt{-1}$.

Equation (11.12) can be exactly inverted to arrive at an expression for the values of the original signal at $\mathbf{N}$ discrete times in terms of the $\mathbf{N}$ values of the discrete Fourier transform.

$$\mathbf{F(m\,\delta t) = \sum_{k=0}^{N-1} G(k\,\delta f)\, e^{2\,\pi\, i\, k\, m/N}} \tag{11.13}$$

where $\mathbf{m = 0}$, $\mathbf{1}$, ..., $\mathbf{(N-1)}$.

Using Eqs. (11.3) and (11.2), we make the replacement

$$\mathbf{\frac{1}{N} = \delta f\, \delta t} \tag{11.14}$$

in Equations (11.12) and (11.13) to get

$$\mathbf{G(f_k) = \frac{1}{N} \sum_{m=0}^{N-1} F(t_m)\, e^{-2\,\pi\, i\, f_k\, t_m}} \tag{11.15}$$

and

$$\mathbf{F(t_m) = \sum_{k=0}^{N-1} G(f_k)\, e^{2\,\pi\, i\, t_m\, f_k}} \tag{11.16}$$

where $\mathbf{f_k = k\,\delta f}$ and $\mathbf{t_m = m\,\delta t}$.

A simpler looking set of transform pairs can be achieved by letting $\mathbf{F_m = F(t_m)}$ and $\mathbf{G_k = G(f_k)}$, in terms of which Equations (11.12) and (11.13) become

$$\mathbf{G_k = \frac{1}{N} \sum_{m=0}^{N-1} F_m\, e^{-2\,\pi\, i\, m\, k/N}} \tag{11.17}$$

and

$$\mathbf{F_m = \sum_{k=0}^{N-1} G_k\, e^{2\,\pi\, i\, k\, m/N}} \tag{11.18}$$

We can use

$$\mathbf{e^{-2\,\pi\, i\, m\, k} = (e^{-2\,\pi\, i\, k})^m = (-1)^m = 1} \tag{11.19}$$

to show that the fast Fourier amplitudes have the periodicity $\mathbf{N}$

$$\mathbf{G_{k+N} = G_k}. \tag{11.20}$$

We can now formally admit negative frequencies by letting $\mathbf{k}$ take on negative integral values, and setting $\mathbf{k = -N/2}$ we then get

$$\mathbf{G_{N/2} = G_{-N/2}}. \tag{11.21}$$

This means that the amplitude corresponding to the "Nyquist frequency"

$$\mathbf{f_{Nyquist}} = \frac{\mathbf{N}}{\mathbf{2}}\,\boldsymbol{\delta}\mathbf{f} \tag{11.22}$$

is the same complex number as the amplitude corresponding to the frequency $-\mathbf{f_{Nyquist}}$.
In the **qfft** package, the function **nyquist** calculates what we call the "Nyquist integer" $\mathbf{k_{Nyquist}}$, which is just

$$\mathbf{k_{Nyquist}} = \frac{\mathbf{N}}{\mathbf{2}}, \tag{11.23}$$

in terms of which

$$\mathbf{f_{Nyquist}} = \mathbf{k_{Nyquist}}\,\boldsymbol{\delta}\mathbf{f} \tag{11.24}$$

Likewise we can show that

$$\mathbf{G_{N/2+1}} = \mathbf{G_{-N/2+1}}. \tag{11.25}$$

which means that the amplitude corresponding to the frequency $(\mathbf{N/2+1})\,\boldsymbol{\delta}\mathbf{f} = \mathbf{f_{Nyquist}} + \boldsymbol{\delta}\mathbf{f}$ is the same complex number as the amplitude corresponding to the frequency $-\mathbf{f_{Nyquist}} + \boldsymbol{\delta}\mathbf{f}$. The only useful part of the spectrum in that contained in the frequency interval between zero and $\mathbf{f_{Nyquist}} - \boldsymbol{\delta}\mathbf{f}$, ie., in the range
$\mathbf{k = 0, 1, 2, ..., (N/2) - 1}$.

In a similar manner we can show that $\mathbf{F_{m+N}} = \mathbf{F_m}$, or that $\mathbf{F(t_m + T)} = \mathbf{F(t_m)}$, where

$$\mathbf{T} = \mathbf{N}\,\boldsymbol{\delta}\mathbf{t} = \frac{\mathbf{N}}{\mathbf{f_s}} = \frac{\mathbf{1}}{\boldsymbol{\delta}\mathbf{f}}, \tag{11.26}$$

so that the fast Fourier amplitudes describe a signal which has the basic inevitable long period $\mathbf{T}$ no matter what other shorter periods (and correspondingly higher frequencies) are also present in the signal. This low frequency, long period property is an artifact of the approximate nature of Equations (11.12) and (11.13).

The fast fourier transform and its inverse should be considered as a distinct type of transform pair rather than as an approximation to either a Fourier series expansion or a Fourier integral expression of a continuous spectrum. The basic idea of the fast Fourier transform is that one has waited long enough for a physical system to "settle down" and the system is then sampled for a certain finite length of time $\mathbf{T_s}$ (we use frequency-time language only for simplicity here, the same ideas apply to wavelength-spatial domain problems).

11.3 Syntax of qfft.mac Functions

```
   FAST FOURIER TRANSFORM UTILITIES
    nyquist (ns,fs)
    sample (expr,var,ns,dvar)
    vf (flist,dvar)
    kg (glist)
    fchop (expr)
    fchop1 (expr,small)
    current_small ()
    setsmall (val)
    spectrum ( glist, nlw, ymax, k1,k2 )


1.  nyquist (ns, fs), given ns, the number of signal
    samples, and fs, the sampling frequency,
    returns the list [dt,knyq,fnyq,df] where dt is the time
    interval between function samples (dt = 1/fs), knyq in the Nyquist
    integer (knyq = ns/2),  fnyq is the Nyquist frequency
    (fnyq = fs/2 = knyq*df), and df is the frequency
    resolution (df = fs/ns) for the output of fft.
    For given ns, the values of df and dt are linked
    by the equation df*dt = 1/ns.
        Thus  nyquist(8,8); returns the list
             [0.125, 4, 4.0, 1.0], and also prints out:

        sampling interval dt =  0.125
        Nyquist integer knyq =  4
        Nyquist freq fnyq =  4.0
        freq resolution df =  1.0

2. sample (expr, var, ns, dvar) constructs a list of ns
   floating point samples F(m*dvar),
   [F(0), F(dvar), F(2*dvar),...,F((ns-1)*dvar) ],
   given the expression expr depending on var.

      sample(cos(t),t,16,1) returns a list of 16 values
      of cos(t) at intervals dt = 1, with list element
      number 1 holding cos(0), element number 2 holding
      cos(dt) = cos(1), etc.

          A signal sample list to be used with fft(flist) should
          have a length ns which is 2 raised to some integer power,
          2^3 = 8, 2^4 = 16,... If your experimental signal sample
          size does not have such a power of 2 length, you should pad
          the sample list with extra zeros.

3. vf ( flist, dvar ), given a list of function samples flist,
    consisting of ns values, and the step size dvar, returns a
        list of the form (if dvar = dt, say),
   [ [0, F(0)],[dt, F(dt)],...,[(ns-1)*dt, F( (ns-1)*dt)]]
   useful for a plot.
    For example, if the length of flist is 8, vf (flist, 1) returns the list
       [ [0, F(0)], [1, F(1)], ... , [7, F(7)] ]
```

```
4.  kg ( glist ) constructs a list of [ k, abs( g(k*df) ) ]
     for k = 0, 1, ..., knyq, which can be used with plot2d.
         knyq is the Nyquist integer, knyq = ns/2,
     where ns is the number of function samples and also the
     length of flist and glist. fchop(abs(glist[j])) is
     used to be able to plot real numbers and set tiny
         numbers to zero.

     glist is the fast Fourier transform list of complex amplitudes
         produced by glist : fft( flist ).

         Using inverse_fft( glist ) should produce flist again to within
         floating point errors. Since floating point errors will also
         introduce tiny imaginary numbers in inverse_fft ( fft (flist) )
         (if flist is real), you can use realpart(...) to recover
         a list of real numbers.

5.  fchop(expr) or fchop(list) sets tiny floating point
          numbers (with package default, less than 10^(-13) ) to zero.

6.  fchop1(expr,small) is used to override the default value of
       the small chop value with your desired value.

      Example: fchop1(s1, 1.0e-3) to set numbers smaller than
       10^(-3) to zero in the expression or list s1.

7.  current_small() returns the current default small chop value.

8.  setsmall(val) allows you to set a new value for the
     current default small chop value; use floating point
     numbers like 2.0e-3 or 2.0E-3.

9.  spectrum (glist, nlw, ymax ) creates a histogram of the frequency
    spectrum implied by glist = fft ( flist ), with line width nlw
        and vertical canvas height ymax. The range of integers
        k is 0 <= k <= knyq. The frequency associated with a given
        line is given by f = k*df, where df is the frequency
        resolution df = fs/ns. ns is the total number of signal samples
        and fs is the sampling frequency: fs = 1/dt (dt is the time
        interval between signal samples).

        spectrum (glist, nlw, ymax, k1, k2) restricts the plot to
        the range k1 <= k <= k2.
```

11.4 The Discrete Fourier Transform Derived via a Numerical Integral Approximation

Review of Trapezoidal Rule

If we let $\mathbf{f_0 = f(a),\ f_1 = f(a+h),\ f_2 = f(a+2\,h)}$, ... and $\mathbf{f_N = f(b) = f(a+N\,h)}$, then the trapezoidal rule approximation is

$$\int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)\,dx} \approx \frac{\mathbf{h}}{\mathbf{2}}\,(\mathbf{f_0 + 2\,f_1 + 2\,f_2 + \cdots + 2\,f_{N-1} + f_N}) \tag{11.27}$$

where $\mathbf{b = a + N\,h}$ defines $\mathbf{h = (b-a)/N}$. If we now specialize to functions such that $\mathbf{f(a) = f(b)}$ then the trapezoidal rule reduces to

$$\int_{\mathbf{a}}^{\mathbf{b}} \mathbf{f(x)\,dx} \approx \mathbf{h}\ (\mathbf{f(a) + f(a+h) + f(a+2\,h) + \cdots + f(b-2\,h) + f(b-h)}) \tag{11.28}$$

with $\mathbf{h = (b-a)/N}$.

If we now make the replacements $\mathbf{x \to t,\ a \to 0,\ b \to T,\ h = (b-a)/N \to T/N = \Delta t}$,
$\mathbf{(b-h) \to T - (T/N) = (N-1)\,\Delta t}$, then

$$\int_{\mathbf{0}}^{\mathbf{T}} \mathbf{f(t)\,dt} \approx \mathbf{(T/N)}\ (\mathbf{f(0) + f(\Delta t) + f(2\,\Delta t) + \cdots + f(\,(N-1)\,\Delta t)}\,) \tag{11.29}$$

or

$$\int_{\mathbf{0}}^{\mathbf{T}} \mathbf{f(t)\,dt} \approx \mathbf{\Delta t} \sum_{\mathbf{m=0}}^{\mathbf{N-1}} \mathbf{f(m\,\Delta t)} \tag{11.30}$$

where $\mathbf{\Delta t = T/N}$ and assuming $\mathbf{f(0) = f(T)}$.

A Path to the Discrete Fourier Transform

If we knew the value of a signal $\mathbf{F(t)}$ at all moments of the interval $\mathbf{0 \le t \le T}$ then we could evaluate the Fourier coefficients

$$\mathbf{C_k} = \frac{\mathbf{1}}{\mathbf{T}} \int_{\mathbf{0}}^{\mathbf{T}} \mathbf{F(t)\,e^{-2\,\pi\,i\,k\,t/T}\,dt} \tag{11.31}$$

($\mathbf{k}$ is an integer) in terms of which the signal could be represented as the sum

$$\mathbf{F(t)} = \sum_{\mathbf{k=-\infty}}^{\mathbf{\infty}} \mathbf{C_k\,e^{2\,\pi\,i\,k\,t/T}} \tag{11.32}$$

which would be a complex form of Fourier series expansion in terms of an infinite number of Fourier coefficients $\mathbf{C_k}$. We have adopted here sign and prefactor conventions which will lead us to Maxima's fast Fourier conventions.

Now assume we only know the signal at $\mathbf{N}$ discrete values $\mathbf{F(m\,\Delta t)}$, where $\mathbf{m = 0, 1, 2}$, ..., $\mathbf{N-1}$, and that $\mathbf{F(t) = f(t+T)}$. We can then approximate the integral in Eq.(11.31) using the trapezoidal approximation expressed by Eq.(11.30).

$$\mathbf{C_k} \approx \frac{\mathbf{1}}{\mathbf{T}}\,\mathbf{(T/N)} \sum_{\mathbf{m=0}}^{\mathbf{N-1}} \mathbf{F(m\,\Delta t)\,e^{-2\,\pi\,i\,k\,m\,\Delta t/T}} \tag{11.33}$$

Defining a "frequency resolution" $\boldsymbol{\delta f}$ given by

$$\boldsymbol{\delta f} = \frac{\mathbf{1}}{\mathbf{T}} = \frac{\mathbf{1}}{\mathbf{\Delta t\,N}} = \frac{\mathbf{f_s}}{\mathbf{N}} \tag{11.34}$$

in which $\mathbf{f_s = 1/\Delta t}$ is the "sampling frequency", and making the replacements $\mathbf{T \to 1/\delta f,\ C_k \to G(k\,\delta f)}$

$$\mathbf{G(k\,\delta f)} = \frac{\mathbf{1}}{\mathbf{N}} \sum_{\mathbf{m=0}}^{\mathbf{N-1}} \mathbf{F(m\,\Delta t)\,e^{-2\,\pi\,i\,(k\,\delta f)\,(m\,\Delta t)}} \tag{11.35}$$

With $\mathbf{f_k = k\,\delta f}$ and $\mathbf{t_m = m\,\Delta t}$, we can write this as

$$\mathbf{G(f_k) = \frac{1}{N}\sum_{m=0}^{N-1} F(t_m)\,e^{-2\,\pi\,i\,f_k\,t_m}} \tag{11.36}$$

which we recognise as the same as Eq.(11.15). With only $\mathbf{N}$ values of $\mathbf{F(t_m)}$ we can only determine $\mathbf{N}$ values of $\mathbf{G(f_k)}$, which by convention we take to be for the values of the integer $\mathbf{k = 0, 1, 2, ..., N - 1}$.

We see from Eq.(11.34) that

$$\mathbf{\delta f\,\Delta t = \frac{1}{N}} \tag{11.37}$$

which allows Eq.(11.35) to be written as

$$\mathbf{G(k\,\delta f) = \frac{1}{N}\sum_{m=0}^{N-1} F(m\,\Delta t)\,e^{-2\,\pi\,i\,k\,m/N}} \tag{11.38}$$

which reproduces our starting point Eq.(11.12) in Sec. 11.2.

The Inverse Discrete Fourier Transform

Although the discrete Fourier transform, Eq.(11.38), is an approximation which gets better as $\mathbf{N}$ increases for fixed $\mathbf{T}$ (or since Eq.(11.34) indicates that the latter condition is equivalent to the ratio $\mathbf{f_s/N}$ being fixed, gets better as $\mathbf{N}$ and $\mathbf{f_s}$ are each increased in the same ratio), the inversion formula involves no further approximations.

Multiplying both sides of Eq.(11.38) by $\mathbf{exp(2\,\pi\,i\,k\,n/N)}$ and then summing both sides over $\mathbf{k}$, and then interchanging the order of the $\mathbf{k}$ and $\mathbf{m}$ summations on the right hand side, results in

$$\mathbf{\sum_{k=0}^{N-1} G(k\,\delta f)\,e^{2\,\pi\,i\,k\,n/N} = \frac{1}{N}\sum_{m=0}^{N-1} F(m\,\Delta t)\sum_{k=0}^{N-1} e^{2\,\pi\,i\,k\,(n-m)/N} = \frac{1}{N}\sum_{m=0}^{N-1} F(m\,\Delta t)\,(N\,\delta_{m,n}) = F(n\,\Delta t)} \tag{11.39}$$

which reproduces our discrete inverse Fourier transform formula Eq.(11.13), since $\mathbf{n}$ is an arbitrary integer.

That the sum over $\mathbf{k}$ is equal to zero if $\mathbf{m \neq n}$ can be seen by letting $\mathbf{l = n - m}$ and using $\mathbf{e^{k\,a} = (e^a)^k}$ and recognising that we have a simple geometric sum. We can also let Maxima confirm this as follows:

```
(%i1) declare([k,l,N],integer)$
(%i2) sum (exp(2*%pi*%i*k*l/N),k,0,N-1),simpsum;
                                 2 %i %pi l
                               %e           - 1
(%o2)                          ---------------
                                 2 %i %pi l
                                 ----------
                                     N
                               %e           - 1
(%i3) %,demoivre;
(%o3)                                  0
```

In the last step the exponentials of complex arguments are converted into their trig function equivalents using

$$\mathbf{e^{i\,\theta} = \cos\,\theta + i\sin\,\theta} \tag{11.40}$$

11.5 Fast Fourier Transform References

We have consulted the treatment of the fast Fourier transform in the following books:

1. **Mathematica for the Sciences, Richard E. Crandall, Addison-Wesley, 1991**
2. **A First Course in Computational Physics, Paul L. Davies, John Wiley, 1994**
3. **Applied Mathematica: Getting Started, Getting it Done, William T. Shaw and Jason Tigg, Addison-Wesley, 1994**