Fourier Analysis Using a TI-Nspire

R. L. Maybach

March 1, 2013

This note describes routines that demonstrate the features and limitations of Fourier analysis and also analyze laboratory data. Topics discussed include:

- effects if including a finite number of terms,
- sampling effects, and
- importance of the choice of the Fourier calculation interval.

Although the functions are written for the TI-Nspire, they can easily be adapted to run on a TI-89 by changing the assign commands (:=) to store commands. However, the TI-Nspire's higher speed and better graphics offer significant advantages.

A Fourier series approximates a periodic function by the sum of its average value plus a set of harmonically-related sinusoids. The lowest frequency (the fundamental) is the reciprocal of the Fourier measurement interval, and the others (the harmonics) are integer multiples of it. The terms of the series (called the Fourier coefficients) are the amplitudes and phases of the sinusoids; however, here we'll express them as complex amplitudes. Frequently, only a few Fourier coefficients need be calculated, and the ready availability of a hand calculator makes it valuable in the laboratory.

This note describes three calculator functions that perform Fourier analysis.

- four – calculates a Fourier series from a calculator function; the name of the function is one of the arguments.
- fourd – calculates a Fourier series from a list of samples of a time waveform; these can be laboratory data.
- fouri – calculates the f(t) represented by a Fourier series: it’s most often used with the TI-Nspire’s graphing features to plot f(t).

It also includes descriptions of two auxiliary functions that are used with the above.

- samp – generates a list of equally-spaced values of a calculator f(t), which can be used with fourd to explore the effects of sampling.
- nser – generates a list of equally-spaced numbers, typically lists of frequencies or sampling times; these are used to graph Fourier spectra or data samples using scatter plots.

Finally, there are three test functions to exercise the above:

- fn1 – a sine wave,
- fn2 – a sine wave that has been full-wave rectified, and
- fn3 – a pulse train with a duty cycle of 0.25.

Using the Functions

The easiest way to understand the use of the above functions is by following some examples.

Example 1 – Sine Wave

The time function is fn1(p,t), a sine wave with a period p. Figure 1 shows its plot with p =1 second.

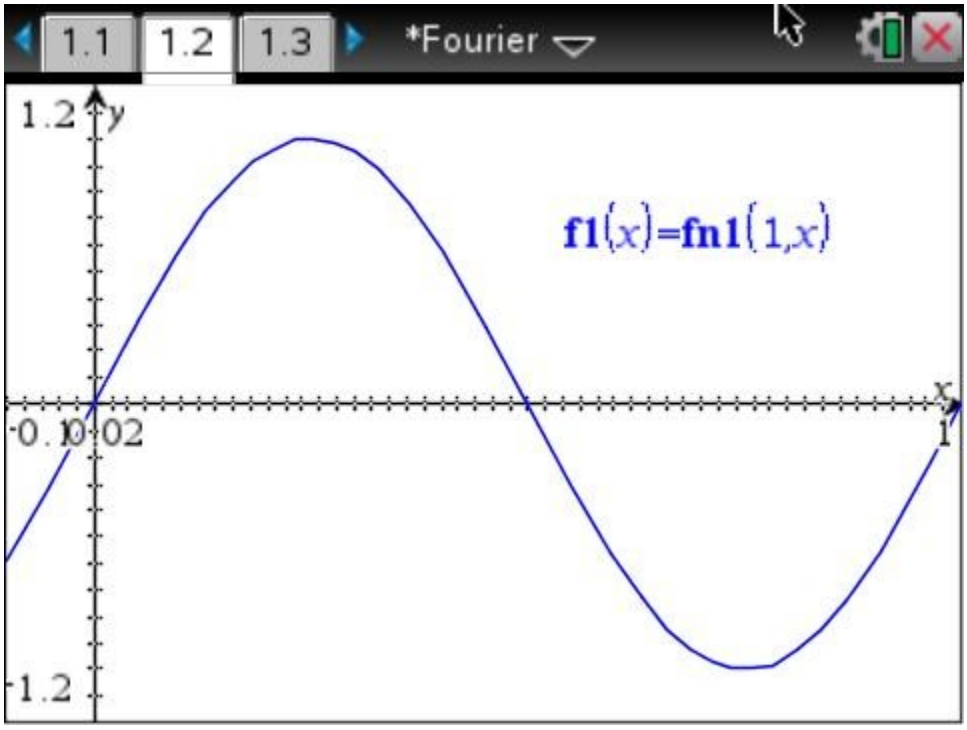

Figure 1. Plot of fn1(1,x)

Calculate the Fourier series with four("fn",p,q,n), where

- fn is the function to be analyzed. Note the quotes, which are essential, since we are passing just the name of the function.
- p is the period of the function to be analyzed. This is passed to fn by four.
- q is the Fourier measurement interval.
- n is the number of coefficients to be calculated.

Figure 2 shows the calculations. The Fourier series is fs1_, which corresponds to a 1-Hz sine wave (generated by fn1 inside four). (The training underscore indicates that fs1_ contains complex numbers.) The Fourier coefficients are calculated over a two-second interval, which produces a list of eight coefficients (for frequencies of 0, 0.5, 1, 1.5, 2, 2.5, 3, and 3.5 Hz). All we need here are their magnitudes, so we'll create a new series, fm1, the absolute values of the items in fs1_. Next, we use nser(del,n) to generate a frequency list, fl1, where

- del is the interval between the points (½ second), and

- n is the number of points (8).

For this case, we want eight points, spaced by 0.5, since our Fourier measurement was two seconds.

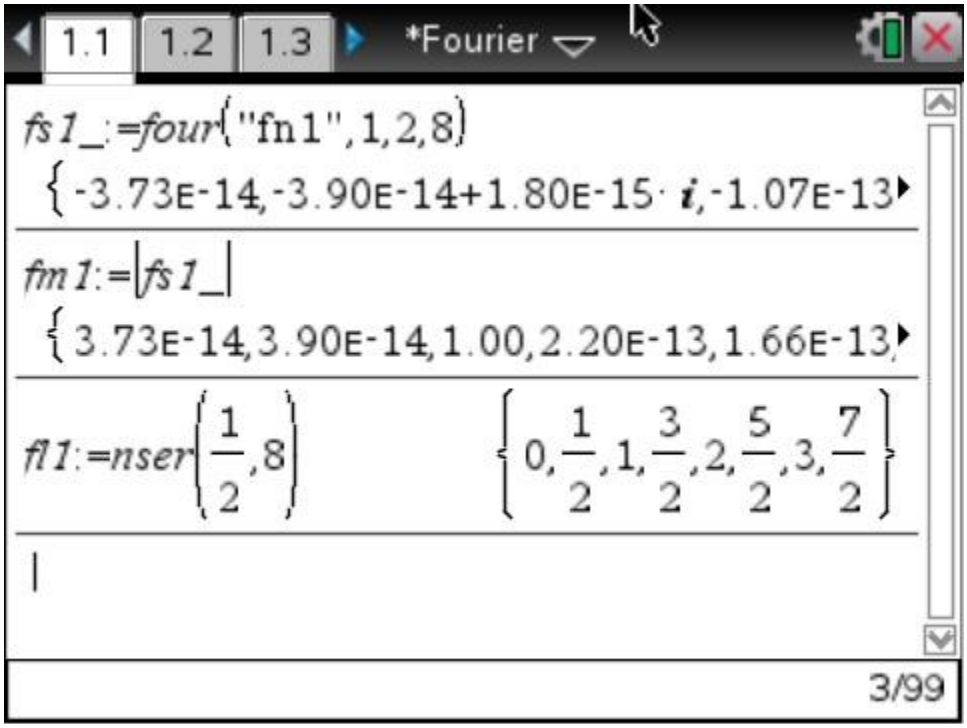

Figure 2. Calculations to find and graph the Fourier series for a sine wave.

We graph this as a scatter plot with x=fl1 (the frequencies) and y=fm1 (the magnitudes of the coefficients); see Figure 3.

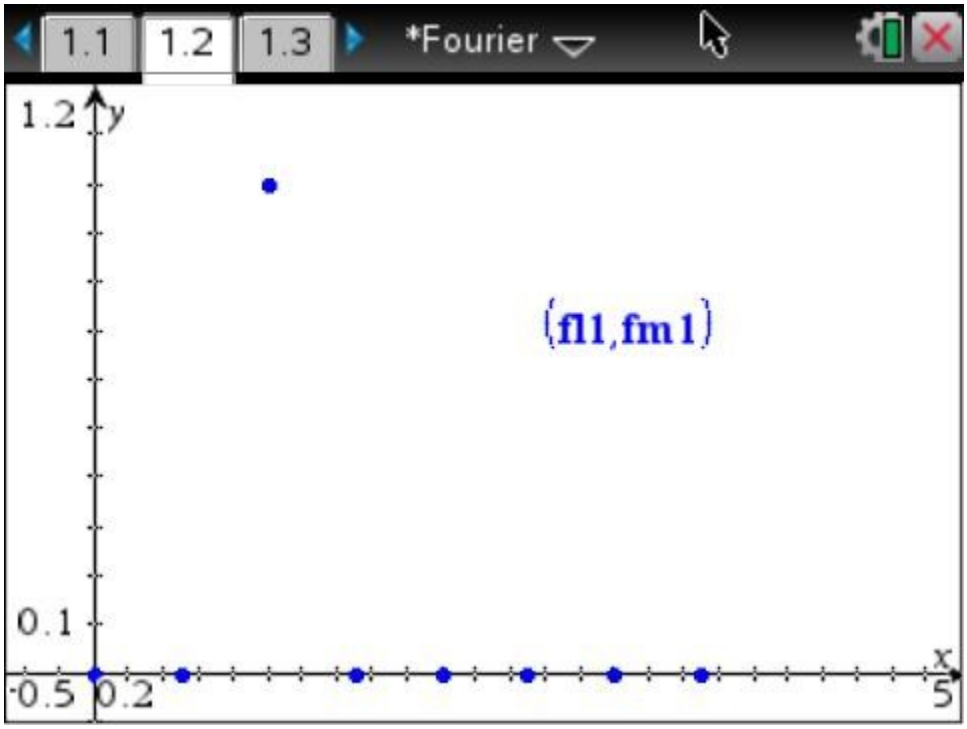

Figure 3. Fourier series magnitudes for a 1-Hz sine wave.

As we expect, the 1-Hz term is one, and all the others are zero.

Example 2 – Rectified Sinusoid

Let's examine a less trivial example, a full-wave rectified sinusoid. We begin by plotting the function `fn2(1,t)`, where the "1" indicates that the period is one second; see Figure 4.

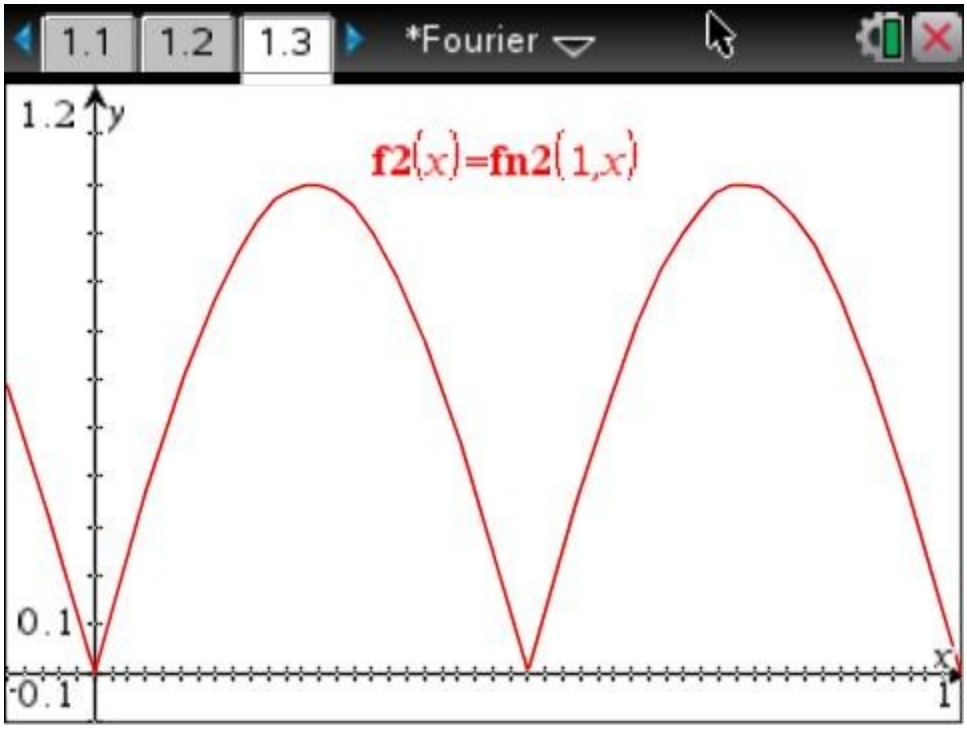

Figure 4. Graph of a rectified sine wave.

We find the Fourier series using the same procedure as before, as shown in Figure 5.

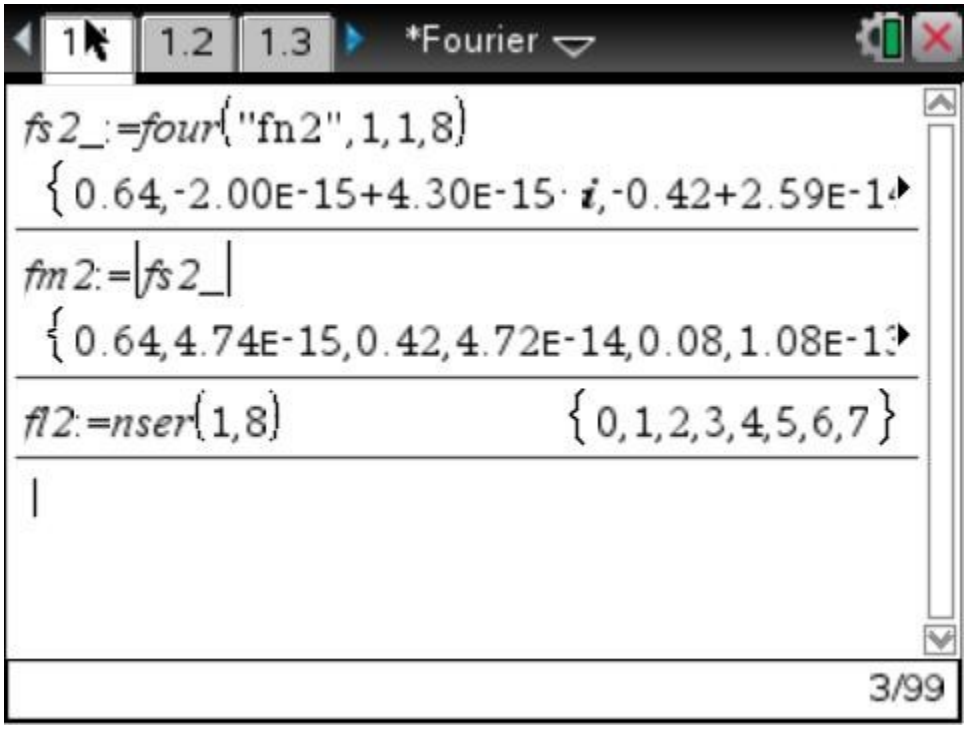

Figure 5. Calculations to find and graph the Fourier series for a rectified sine wave.

Note that the last three arguments for four("fn",p,q,n) are 1, 1, and 8, showing that the period of fn2 is one second, that the Fourier calculation also extends over one second, and we want eight coefficients. As a result, the harmonics are separated by 1 Hz. As before, we'll use a scatter plots to graph fm2; see Figure 6.

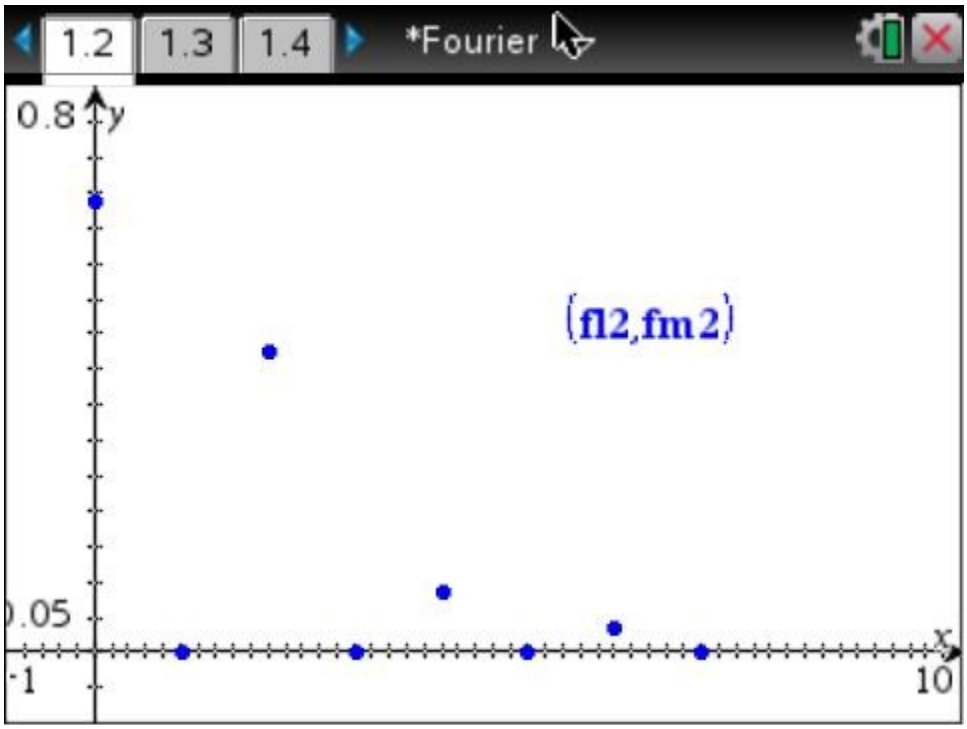

Figure 6. Fourier series magnitudes for a 1-Hz rectified sine wave.

At this point, we could link fl2, and fm2 to columns in a spreadsheet to make calculations using these results.

Finally, we can fouri(c_,q,x) to plot the waveform generated by the Fourier series, where

- c_ is the list of (complex) Fourier coefficients (fs2_ here),
- q is the Fourier calculation interval (1 second in this case), and
- x is the independent variable.

Figure 7 shows these, with the graphing commands needed shown on the plot. To the scale shown, the agreement is good, although the error is noticeable near the x-axis.

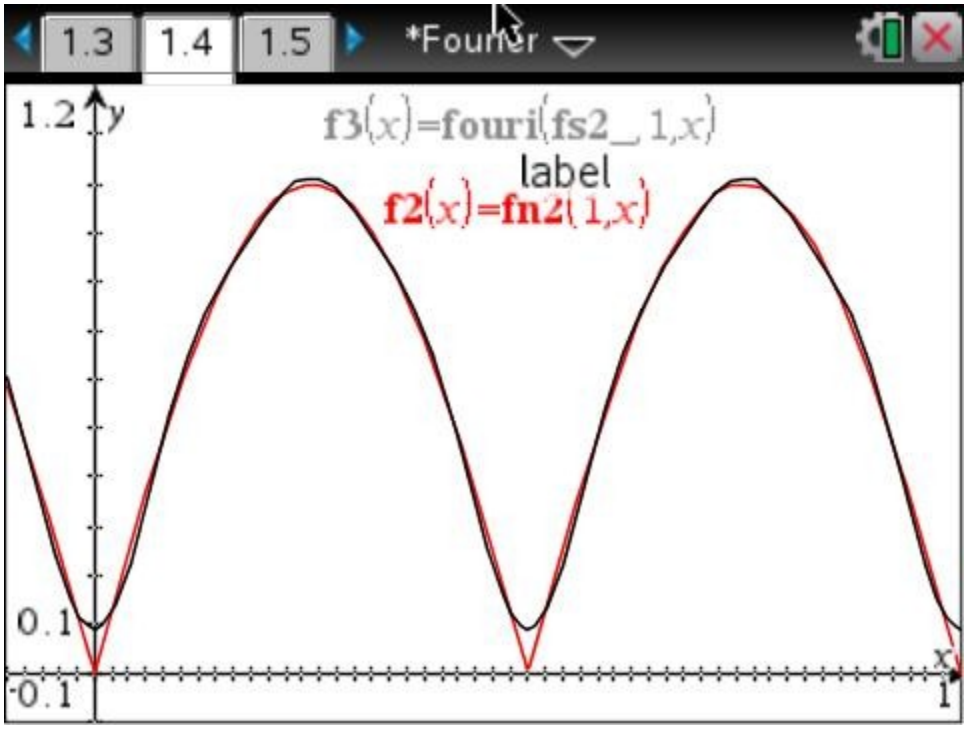

Figure 7. Rectified sine wave generated from an eight-term Fourier series.

Example 3 – Pulse Train

We'll use the function fn3, which is a pulse train with a duty cycle of 25 per cent. Figure 8 shows the calculations for the Fourier series, fs3_, its magnitudes, fm3, and their frequencies, fl3.

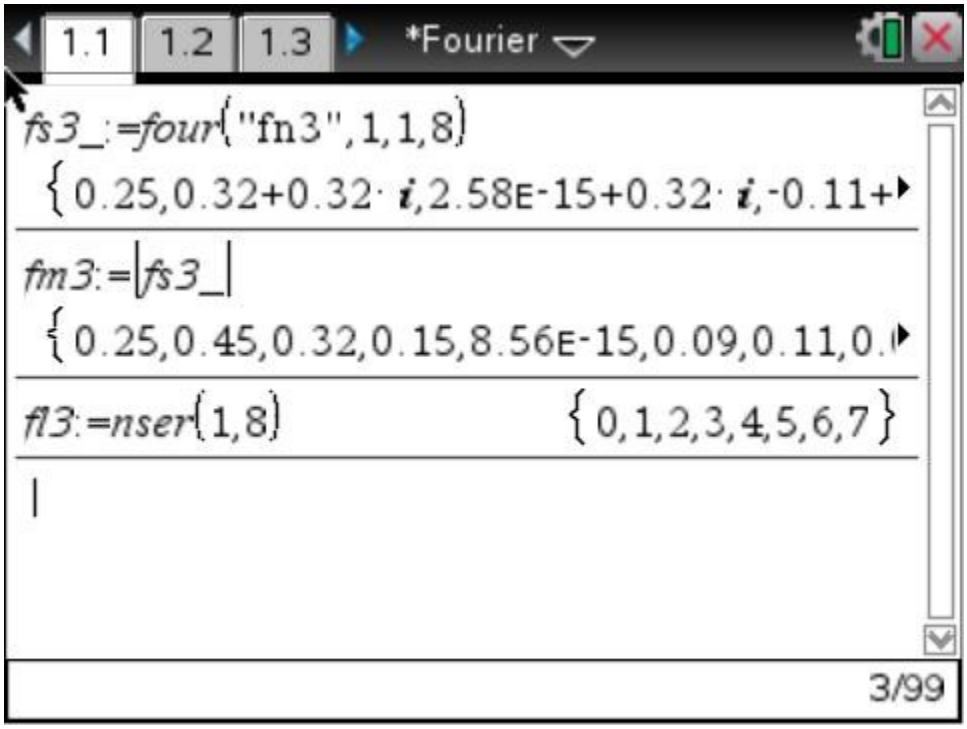

Figure 8. Calculations to find and graph the Fourier series for a pulse train.

The periods of both the pulse train and the Fourier calculation windows are one second, and the Fourier series contains eight coefficients.

Figure 9 shows the magnitudes of the Fourier components, and Figure 10 the original signal and its Fourier approximation. In this case, the approximation is not particularly good; we would need many more components to improve it significantly.

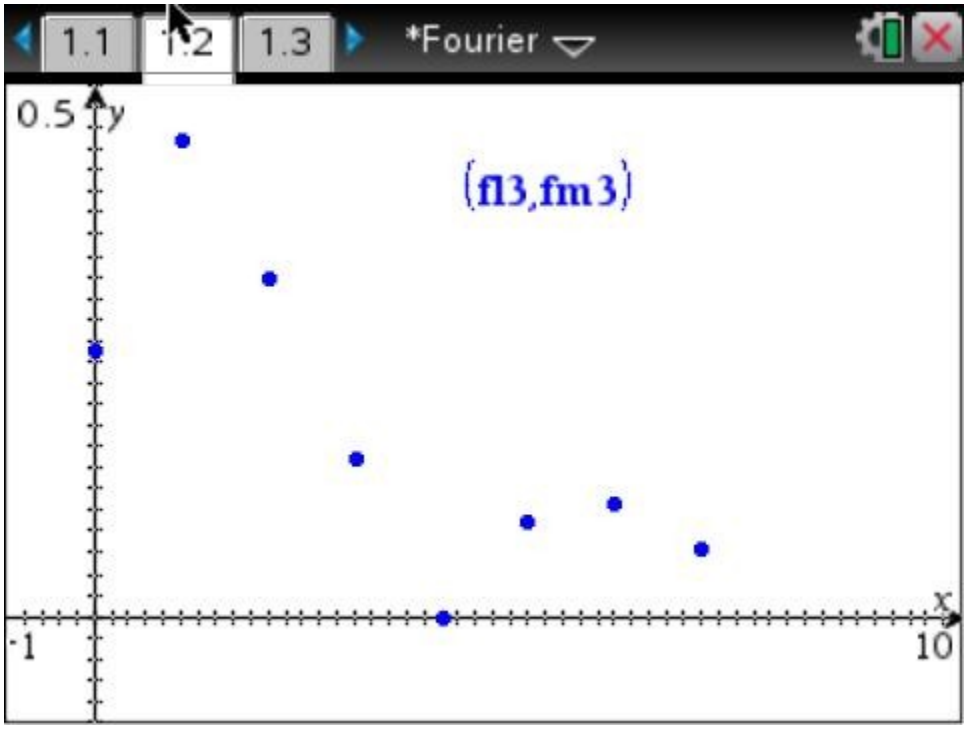

Figure 9. Fourier series for a pulse train.

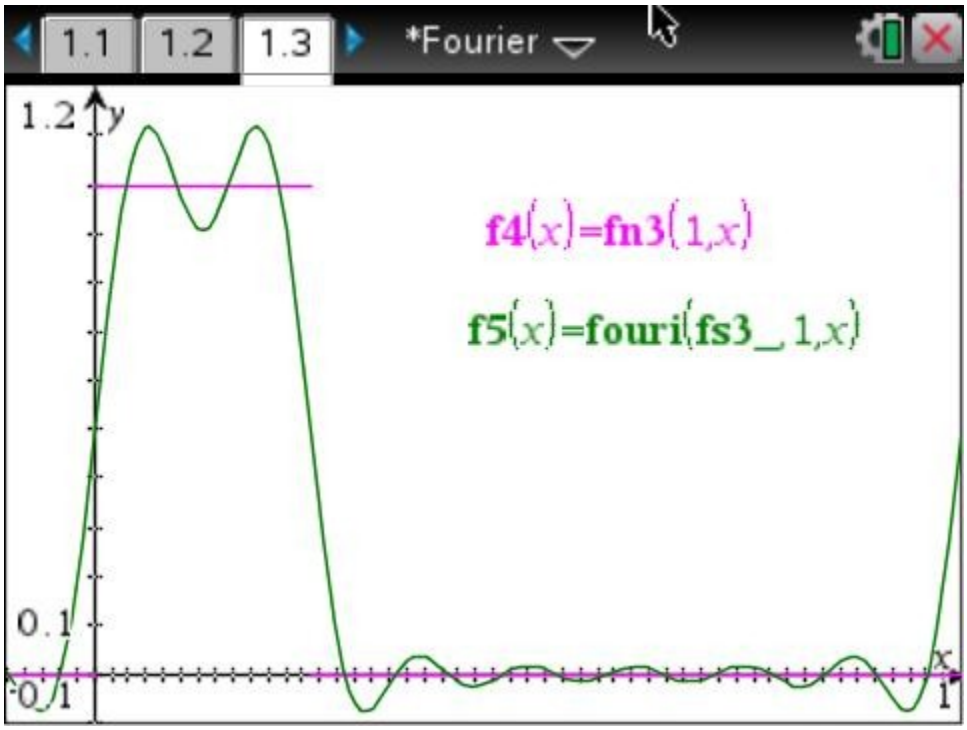

Figure 10. Regenerated and original signals for a pulse train.

Example 4 – Sampled Pulse Train

Here, we'll again use the pulse train fn3, but we'll sample it with samp("fn", p,q,n), where:

- fn is the function to be analyzed (fn3 here);
- p is the period of the function to be analyzed (1 second);
- q is the sampling interval (also 1 second); and
- n is the number of samples taken (20).

Figure 11 shows the calculations to generate sl4, the list of 20 samples over the 1-second period of fn3 and to create list of sample times, tl4. Figure 12 plots the samples and the original function. (Note that if we look only at the samples, the pulse appears to be shifted to the left.)

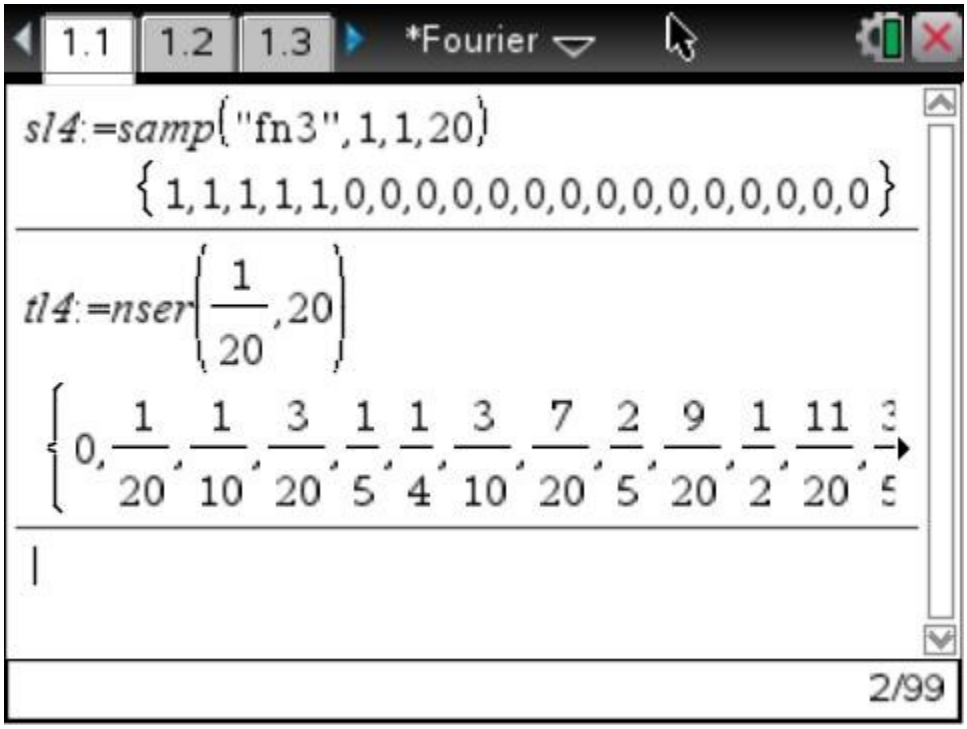

Figure 11. Calculations to sample fn4 and create a list of sample times.

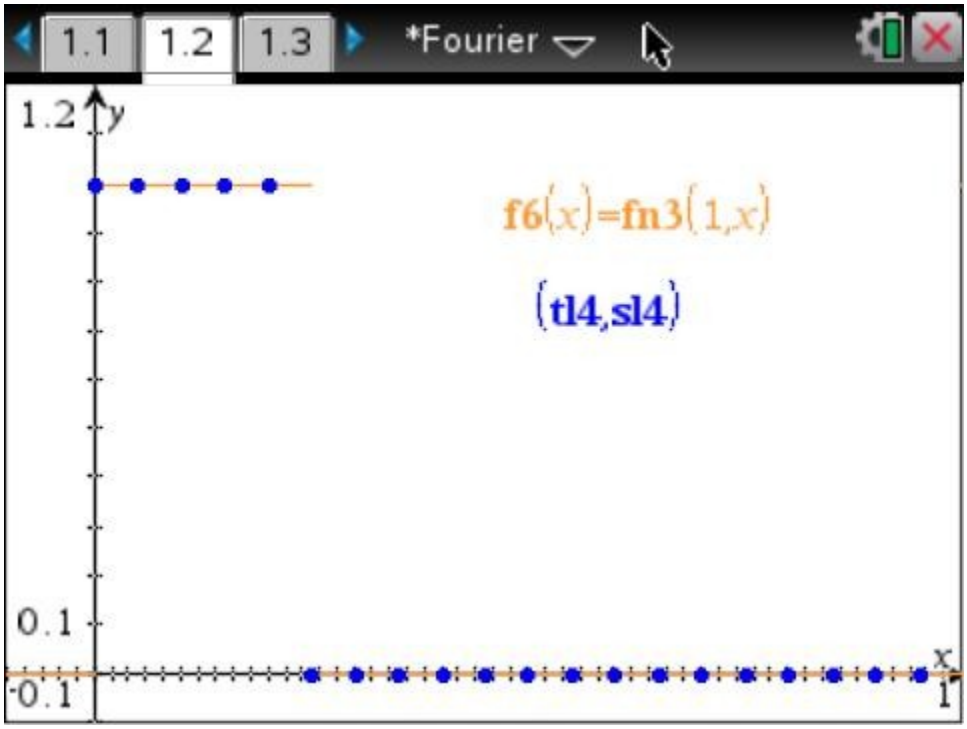

Figure 12. Graph of fn3(1,x) and its list of samples, sl4.

Figure 13 shows the calculation to find the Fourier coefficients, fs4_, of these samples, using the function fourd(a,n), where

- a is the list of samples (fn4sf), and
- n is the number of coefficients (8) to be computed.

It also shows the calculation of the spectrum (the magnitudes of the coefficients), fm4, and a list of their associated frequencies, fl4.

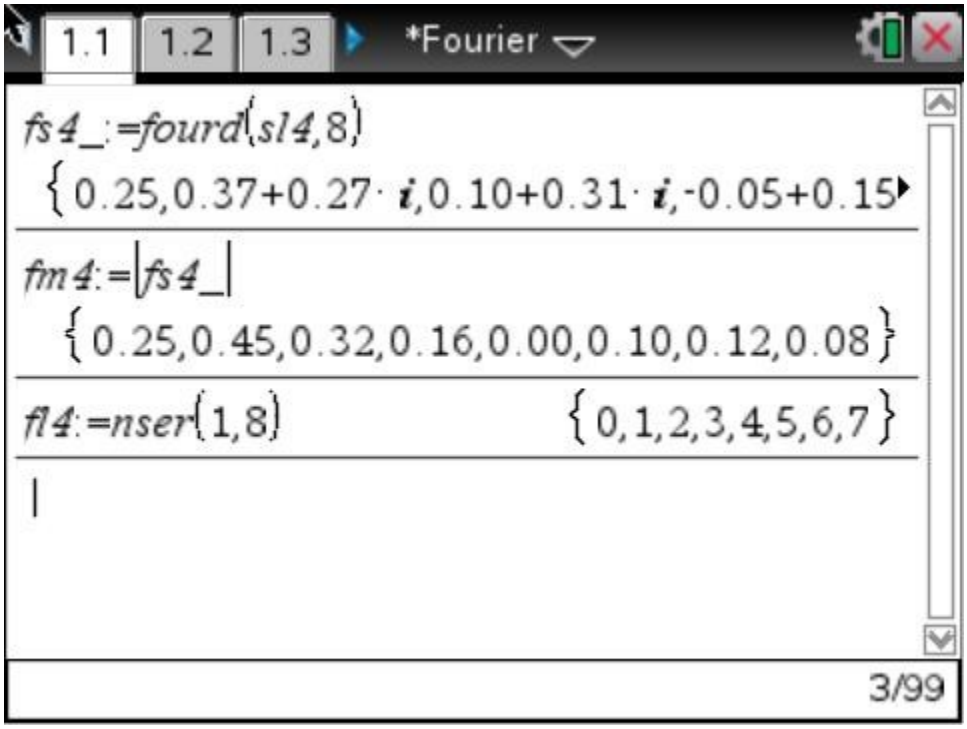

Figure 13. Calculation to find and plot the Fourier series for a pulse train.

In order to calculate a Fourier coefficient, sampling theory requires that more than two samples be taken during its period. In other words, the sampling rate must be greater than twice the frequency of the highest Fourier coefficient. Although four will accept any value of n, the number of coefficients it returns won't exceed half the number of data points. For example, since sl4 contains 20 samples, fourd will return at most 10 coefficients, regardless of how large n is.

As usual, we can graph the spectrum using a scatter plot, shown in Figure 14. Comparing this with Figure 9, we see that sampling has not greatly affected the spectrum.

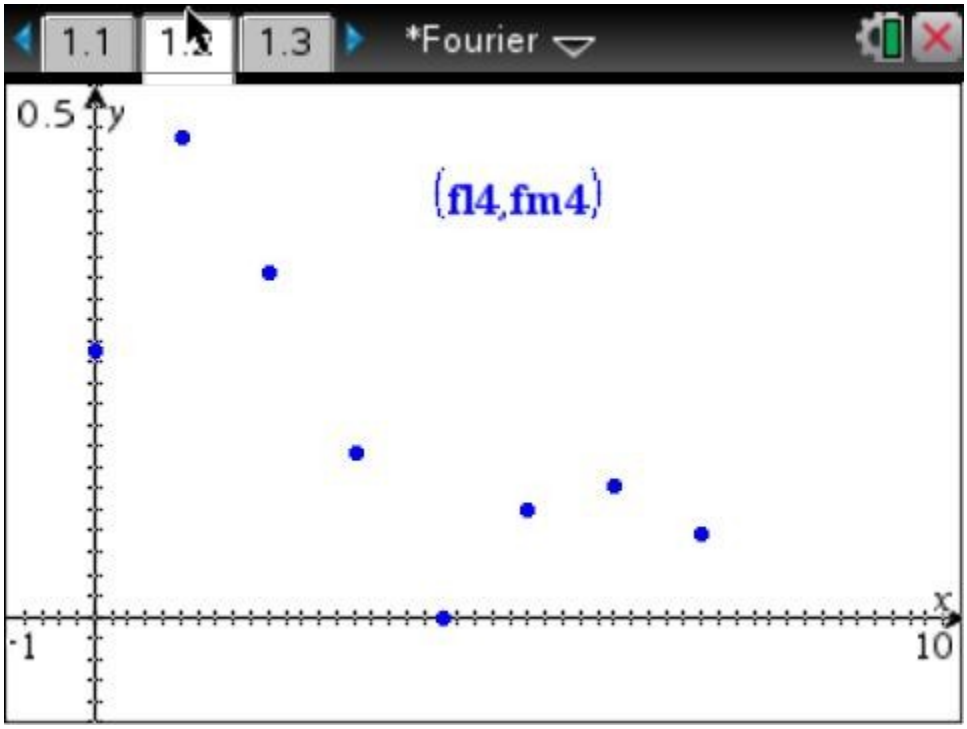

Figure 14. Fourier series for a sampled pulse train.

Figure 15 shows the function reconstructed from the Fourier series. Note that it is shifted to the left from fn4, an effect caused by way the samples are distributed along the signal. This is always problematic when the function has discontinuities. Note also that the waveform is distributed symmetrically around the sample points. For comparison, the graph also shows the waveform reconstructed from the original (continuous) fn3. Except for the time shift, the two are quite close.

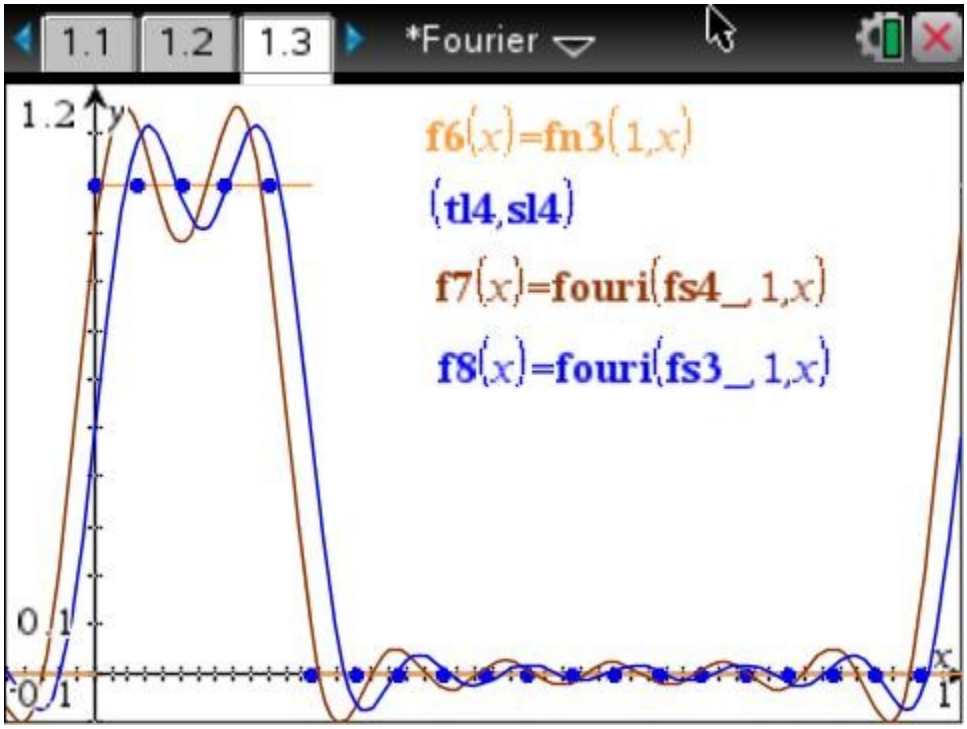

Figure 15. Signals generated Fourier series calculated from sampled and continuous data.

Example 5 – Choice of Fourier Measurement Interval

We have already discussed that the spacing between the Fourier frequencies is the reciprocal of the length of the Fourier measurement interval. Here we look at what happens when the Fourier measurement interval is not an integral multiple of the function period. Figure 16 shows the calculations where the function is a sinusoid with a period of 1.6 second, and the Fourier measurement interval is 2 seconds.

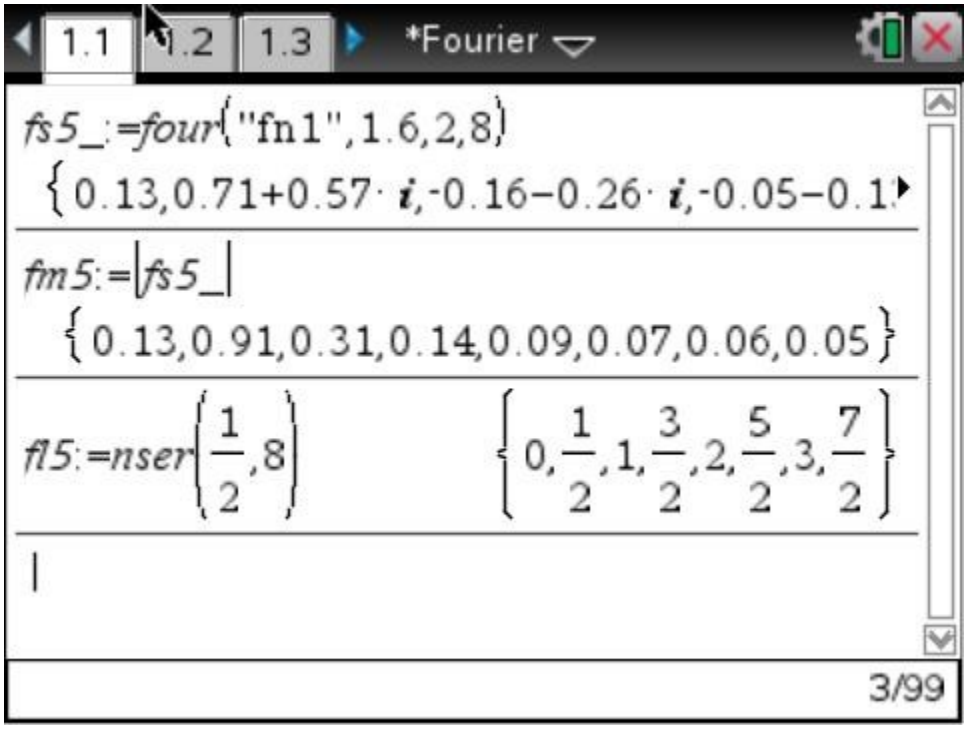

Figure 16. Calculations for Fourier series for a sine wave with a 1.6-second period using a 2-second Fourier interval.

Figure 17 shows the resulting spectrum, which is far different from the correct one shown in Figure 5.

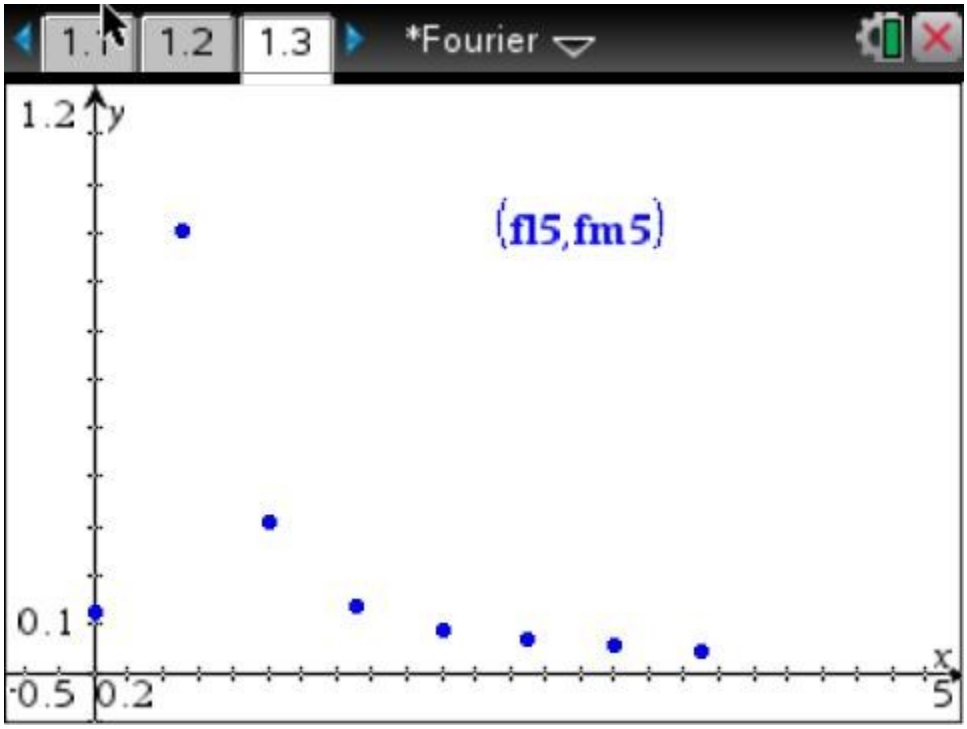

Figure 17. Fourier series for a sine wave with a 1.6-second period using a 2-second Fourier interval.

Figure 18 shows both the original f(t) and the f(t) computed from the Fourier series.

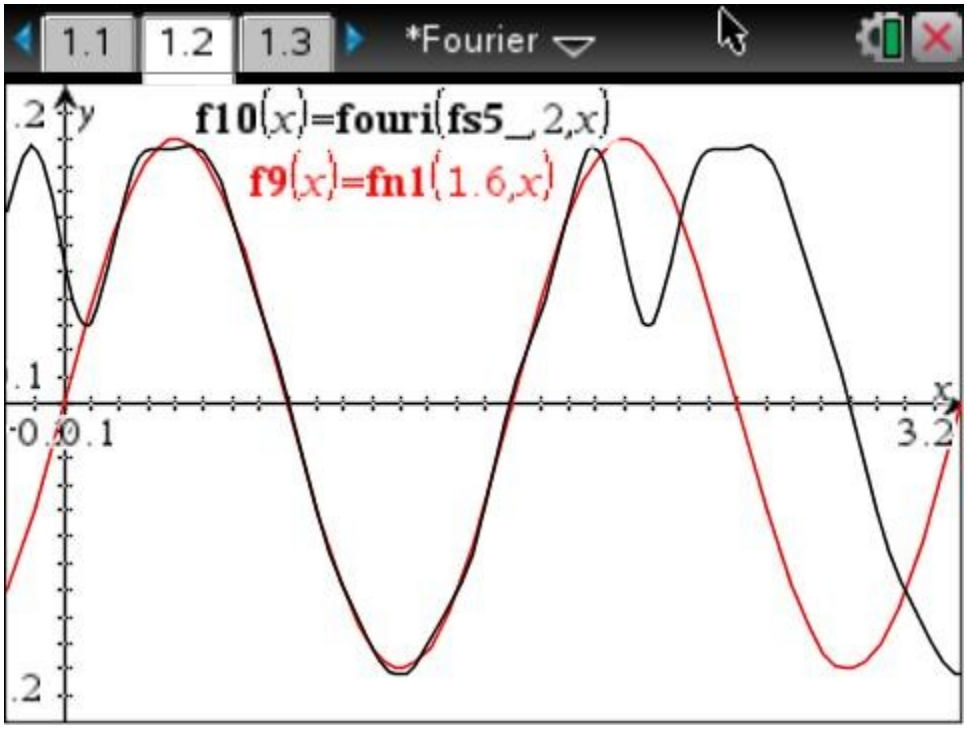

Figure 18. Original and calculated time function.

They are close only within the 2-second Fourier measurement interval; beyond that they diverge dramatically.

Background

Any periodic waveform can be expressed as a sum of sinusoids. (Here, we've followed the engineering convention of using j for the square root of -1.)

$$f(t) = C_0 + \sum_{k=1}^{\infty} |C_k| \cos\left(2\pi kt/T - \angle C_k\right)$$

$$C_0 = \frac{1}{T}\int_0^T f(t)\,dt$$

$$C_k = \frac{2}{T}\int_0^T f(t)e^{j2\pi kt/T}\,dt$$

where

k = 1, 2, 3, … ,

$|C_k|$ denotes the magnitude of C_k, and

$\angle C_k$ denotes its angle.

Summing a series with an infinite number of terms is not practical, which means that we always use a finite number and our series can only approximate the function.

For sampled data, the integrals are replaced by summations.

$$C_0 = \frac{1}{T}\sum_{m=0}^{k} f(m\times\Delta T)\Delta T$$

$$C_k = \frac{2}{T}\sum_{m=0}^{k} f(m\times\Delta T)e^{j2\pi\times n\times m\times\Delta T/T}\Delta T$$

where $\Delta T = \frac{T}{k}$.

When we make the substitution for ΔT and simplify, we obtain

$$C_0 = \frac{1}{k}\sum_{m=0}^{k} f_m$$

$$C_k = \frac{2}{k}\sum_{m=0}^{k} f_m e^{j2\pi\times n\times m/k}$$

where f_m is the m^{th} sample value. Both the continuous and the sampled-data cases result in Fourier series, and the process to generate a tine function is the same in each case.

Calculating the Coefficients

With this background, we can write the calculator functions almost by inspection. For the continuous case, the result is the function four.

```
Define four(fn,p,q,n)=
Func
:©Fourier series for fn(p,t), period q, n terms
:Local b,j,s_
:If getType(#fn)≠"FUNC" Then
:  Return "fn not a function"
:ElseIf getMode(2)≠1 Then
:  Return "Angle mode not radian"
:Else
:  s_:=newList(n)
:  s_[1]:= nInt(#fn(p,t),t,0,q)/q
:  For j,1,n-1
:    b:=(2*π*j/q)
:    s_[j+1]:=(2*nInt(#fn(p,t)*e^(i*b*t),t,0,q)/q)
:  EndFor
:  Return s_
:EndIf
:EndFunc
```

The two characters **e^** denote the exponential function e^x. The ***i*** symbol is not the alphanumeric i, but the square root of -1, which is available when you press the ***π*** key.

The function is called with four arguments:

- fn – a text string containing the name of the function for which the coefficients are desired,
- p – the period of the function,
- q – the Fourier measurement interval (T in the discussion above), and
- n – the number of coefficients.

Note the indexing s[j+1], which results because the Fourier coefficients are numbered 0,1,2, …, while TI-Nspire lists are numbered 1,2,3, … . This occurs in several other functions.

The function makes two elementary error checks. It returns an error message if the angle mode is not RADIAN or if it can't find the function fn. Without the first error check, if the angle mode is DEGREE, the function **e^( )** fails with a cryptic error message. Even worse, without the second error check, if fn doesn't exist, the function nInt( ) will lock up the calculator. The # causes what TI calls *indirection*, and it converts a string into a variable. If the function were called with an fn of "fn1", then in the program #fn(t,p) would be converted to fn1(t,p). The function fn1(t,p) must reside in the directory from which the function four is called, but not necessarily in the directory where four resides.

The function fouri calculates the f(t) corresponding to a Fourier series for a particular value of t. Its arguments are:

- c_ – a list containing the Fourier coefficients, $\{C_0, C_1, C_2, \cdots, C_n\}$. That the variable's name ends in an underscore identifies it as a complex variable,

- p – the period of f(t), and
- t – the time for which f(t) is to be evaluated.

It is most often used with the TI-Nspire's graphing feature to see an entire f(t).

```
Define fouri(c_,q,t)=
Func
:©fouri(c_,q,t) f(t) from c_{c0,c1, ....},q=((1)/(Δf))
:Local j,f
:If getMode(2)≠1 Then
:  Return "Angle mode not radian"
:Else
:  f:=c_[1]
:  For j,1,dim(c_)-1
:    f:=f+abs(c_[j+1]*cos(2*π*j*t/q)-angle(c_[j+1]))
:  EndFor
:  Return f
:EndIf
:EndFunc
```

The function makes one elementary error check. It returns an error message if the angle mode is not RADIAN. (Without this, if the angle mode is DEGREE, the function ***e***^( ) fails with a cryptic error message.)

For the sampled-data case, the result is the function fourd, essentially four with numerical integration replaced by summation.

```
Define fourd(a,n)=
Func
:©fourd(a) Find c_[1] ... c_[n] for list a
:Local b, k,m,c_
:If getType(a)≠"LIST" Then
:  Return "First arg not a list"
:ElseIf getMode(2)≠1 Then
:  Return "Angle mode not radian"
:Else
:  m:=dim(a)
:  n:=min(n,m/2)
:  c_:=newList(n)
:  c_[1]:=∑(a[j+1],j,0,m-1)
:  For k,1,n-1
:    b:=(2*π*j/m)
:    c_[k+1]:=2*∑(a[j+1]*e^(i*2*π*j*k/m),j,0,m-1)
:  EndFor
:  Return (c_/m)*1.
:EndIf
:EndFunc
```

The function is called with two arguments:

- a – list of data points, taken at evenly-space time intervals, for which the coefficients are desired, and
- n – the number of coefficients.

There are two elementary error checks. It returns an error message if the angle mode is not RADIAN or if a is not a list.

The function samp("fn",p,q,n) has the arguments

fn - a text string containing the name of the function for which the samples are desired,
p - the period of the function,
q - the sampling interval, and
n - the number of coefficients.

```
Define samp(fn,p,q,n)=
Func
:Local i,s
:If getType(#fn)≠"FUNC" Then
:  Return "fn not a function"
:Else
:  s:=newList(n)
:  For i,0,n-1
:    s[i+1]:=#fn(p,q*i/n)
:  EndFor
:  Return s
:EndIf
:EndFunc
```

The function nser(del,n) generates a list of equally-spaced numbers. Its arguments are

- del – the interval between the numbers and
- n – the number of items.

```
Define nser(del,n)=
Func
:©Return {0,del,2*del, 3*del, ... ,(n-1)*del}
:Local i,s
:s:=newList(n)
:For i,1,n-1
:  s[i+1]:=s[i]+del
:EndFor
:Return s
:EndFunc
```

The following are the time functions used to demonstrate the functions discussed above. Note the use of the mod function in fn3 to make the function periodic.

```
Define fn1(p,t)=
Func
:©sinusoid with period p
:Return sin(2*π*t/p)
:EndFunc

Define fn2(p,t)=
Func
:©Full-wave rectified sinusoid with period p
:Local s
:s:=sin(2*π*t/p)
:Return when(s<0,−s,s)
:EndFunc

Define fn3(p,t)=
Func
:©Quarter-period pulse with period p
:Return when(mod(t,p)<(p/4),1,0)
:EndFunc
```