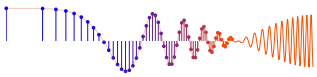


Signal Processing First

# Lab 13: Numerical Evaluation of Fourier Series

**Pre-Lab and Warm-Up:** You should read at least the Pre-Lab and Warm-up sections of this lab assignment and go over all exercises in the Pre-Lab section before going to your assigned lab session.

**Verification:** The Warm-up section of each lab must be completed **during your assigned Lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time**. One of the laboratory instructors must verify the appropriate steps by signing on the **Instructor Verification** line. When you have completed a step that requires verification, simply demonstrate the step to the TA or instructor. Turn in the completed verification sheet to your TA when you leave the lab.

**Lab Report:** It is only necessary to turn in a report on Section 4 with graphs and explanations. You are asked to **label** the axes of your plots and include a title for every plot. In order to keep track of plots, include your plot *inlined* within your report. If you are unsure about what is expected, ask the TA who will grade your report.

## 1 Introduction & Objective

The goal of the laboratory project is to show how to calculate the Fourier Series coefficients, $\{a_k\}$, without doing the integrals by hand. Instead, we will use MATLAB's numerical integration capability to evaluate the integrals numerically. Another approach would be to use a symbolic algebra package such as *Mathematica* or Maple to derive formulas for the Fourier Series coefficients.

### 1.1 Background: Fourier Series Analysis and Synthesis

The Fourier Series representation applies to periodic signals. The Fourier synthesis equation for a periodic signal $x(t) = x(t + T_0)$ is

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{jk\omega_0 t}, \tag{1}$$

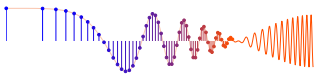
where $\omega_0 = 2\pi/T_0$ is the *fundamental* frequency. To determine the Fourier series coefficients from a time-domain formula for the signal over one period, we must evaluate the *analysis* integral for every integer value of $k$:

$$a_k = \frac{1}{T_0} \int_0^{T_0} x(t) e^{-jk\omega_0 t} dt \tag{2}$$

where $T_0 = 2\pi/\omega_0$ is the *fundamental* period. If necessary, we can evaluate the analysis integral over any period; in (2) the choice $[0, T_0]$ is sometimes a convenient one, but integrating over the interval $[-\frac{1}{2}T_0, \frac{1}{2}T_0]$ would also give exactly the same answer.

## 2 Pre-Lab

In this lab, we will use a feature of MATLAB that allows you to evaluate integrals by numerical approximation. MATLAB has several numerical integration functions, such as `quad()` or `quad8()` that are very powerful methods for evaluating integrals of functions specified by formulas. The "formula" for the integrand must be written as a MATLAB function. Then the numerical integration M-file will adaptively figure



out the best way to approximate the integral with a sum. We can use these functions to evaluate Fourier Series integrals and then we will plot the resulting Fourier Series coefficients $\{a_k\}$ to display the spectrum.

## 2.1 Numerical Integration

The basic idea of numerical integration is to approximate an integral with a sum, and the simplest way to do this is with the Riemann sum:

$$\int_a^b f(t)dt \approx \sum_{n=1}^{L} f(t_n)\left(\frac{b-a}{L}\right) \tag{3}$$

where the "sampling points" $t_n$ are $t_n = a + (n - \frac{1}{2})\Delta$, and $\Delta = \frac{b-a}{L}$. The parameter $\Delta$ is actually the width of subintervals defined by breaking the interval $[a, b]$ into $L$ equal sized subintervals. The sampling points are taken to be the *mid-points* of those subintervals.[1]

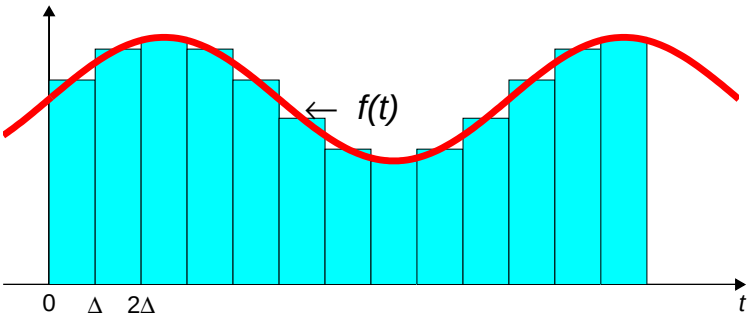



Figure 1: Approximating the area under $f(t)$ with the area of many narrow triangles.

The theory of Riemann integration (in calculus) tells us that taking the limit as $L \to \infty$ will make the sum on the right converge to the integral on the left in (3). Furthermore, if we think of the integral as calculating area, then the Riemann sum is approximating the area with the sum of areas of many rectangles whose width is $\Delta$ and whose height is $f(t_n)$. We could get a better approximation to the area if we used trapezoids instead of rectangles to compute the area under the curve $f(t)$. Carrying this idea to the next step, we can use polynomial approximations to $f(t)$ to get even more accurate area approximations. This is what the MATLAB functions `quad`, `quadl` and `quad8` do. Consult `help quadl` or `help quad8` for more information.

## 2.2 Integrate a Simple Function with MATLAB

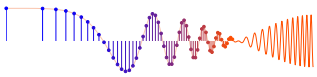
Suppose that we want to calculate the value of the definite integral[2] $\int_0^5 \cos(\pi t)dt = ?$
In MATLAB this can only be done as a numerical approximation with functions such as `quad8`, or `quadl`. In version 6, you will receive a warning that `quad8` is obsolete, but the function still works correctly. In version 5.x, `quad8` exists, but not `quadl`.

### 2.2.1 Numerical Integration with `quad8`

In order to use `quad8('f',a,b)` we must pass three arguments: the function definition (of the integrand), and also the lower and upper limits of integration. The limits of integration are scalar constants, so the tricky

[1] We could define $t_n$ to be at one of the endpoints of the interval, and the same results would be obtained (in the limit) for approximating the integral with a sum.

[2] This easy example was chosen so that you can check the result by hand.



part is passing the function definition. There are two ways to do this: (1) write a separate MATLAB function to define the integrand and use the name in `quad8` , or (2) use the MATLAB function `inline()` to define the entire function via one string.

In MATLAB it seems reasonable that we might be able to define the first argument in `quad8` by writing `'cos(pi*t)'` as a string.[3] However, this won't work with `quad8` unless we use the `inline()` function. (See Section 2.5 below.) The alternative is to write a separate M-file that sets up the integrand as a function that can be evaluated. For example, we might create an auxiliary M-file called `fcos.m` by writing the following simple function:

```
function out = fcos(t)
%FCOS   function definition of a cosine used in an integral
%       the input t can be a vector
%
out = cos(pi*t);
```

Once we have the `fcos()` function stored in the file `fcos.m`, we can perform the integration by using the **name** of the function in the call to `quad8()`:

```
quad8( 'fcos', 0, 5 )
```

You should run this example and verify that it gives the correct answer (because you can do the integral in your head). Notice that MATLAB gives an answer that is correct to the inherent numerical precision of double-precision floating point which is usually about 15 digits of accuracy.

(a) Modify the example above to do the integral of $e^{-|t|}$ from $t = -2$ to $t = 2$. This requires that you write a new auxiliary function M-file to generate "e to the minus absolute value of t."

(b) Verify by hand that the correct value of the definite integral in part (a) is 1.7293.

### 2.2.2 Numerical Integration with `quadl` (Version 6 only)

Starting with version 6 of MATLAB, the function `quad8` is being phased out in favor of one called `quadl`. The function calls are basically the same, except that `quadl` supports two additional ways to define the integrand. First of all, if the M-file name is used, it should be preceded with an "at" symbol, i.e., `quadl(@fcos,0,5)` for the example above. Second, if a text string is used to define the function, it no longer has to be wrapped inside of the `inline()` function, i.e., `quadl('cos(pi*t)',0,5)` works.

## 2.3 Plotting with `fplot`

A useful plotting function when you have a function definition programmed into an M-file is `fplot()`. The syntax for `fplot` is *nearly* the same as `quad8`, except that the range must be given as a two-element vector, not as two separate arguments. From version 5 to 6, the method of calling the function definition has been enhanced from the style required by `quad8` to that used in `quadl`.[4]

Use `fplot()` to plot the function $e^{-|t|}$ over the range $-2 \leq t \leq 2$.

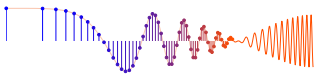
## 2.4 Function Evaluation with `feval`

Both `fplot` and `quad8` are based on MATLAB's evaluation function called `feval`, which can obtain values of a function defined by an M-file. For example, a set of values of the cosine function defined above can be obtained via:

```
feval('fcos',0:0.25:2)
```

[3] This behavior has been corrected in version 6 with the `quadl` function.

[4] Actually, the string input was allowed in version 5.3 where the following is valid: `fplot('cos(pi*t)',[0,5])`.



### 2.5 Inline Functions

We can eliminate the auxiliary M-file if we exploit MATLAB's capability to define the function as a string. This only works when the function is simple and can be expressed as a "one-liner." In version 5, it is necessary to use an "in-line" function definition as a wrapper around the string.

Consult help on `inline`

For the cosine example above, here is the method:

```
quad8( inline('cos(pi*t)'), 0, 5)
```

Likewise, you could call `fplot()` with an inline function definition.

(a) Make a plot of the signal $f(t) = 2 - |t|$ over the range $-1 \leq t \leq 1$ by using `fplot` and `inline`.

(b) Evaluate the definite integral of $f(t) = 2 - |t|$ over the range $-1 \leq t \leq 1$ by using one call to `quad8` and `inline`. Verify by hand that MATLAB gave the correct answer.

## 3 Warmup

In the warm-up you will use `quad8` (or `quadl`) and `fplot` to do more complicated integrals, such as those needed in Fourier Series analysis (2) to extract the Fourier coefficients, $a_k$ .

### 3.1 Piecewise Definition of a Signal

Many signals are defined by giving several cases that define pieces of the overall signal. For example, the square wave is one for part of its period and zero for the rest. There are a couple of ways of dealing with cases when writing the auxiliary M-file needed for `fplot` or `quad8`. The tricky part is making sure that the auxiliary function can handle a vector input. Pay careful attention to the warning in the help for `quad8` that states:

```
Q = QUAD8('F',A,B) approximates the integral of F(X) from A to B...  'F'
is a string containing the name of the function.  The function must return
a vector of output values if given a vector of input values.
```

In other words, you must have a vector output if called by `quad8`.[5] It helps to vectorize the cases; perhaps by using the `find` function or with vectorized logical tests.

(a) Consider the problem of plotting the triangle wave of Fig. 2 In order to define the waveform so that you can plot it over *any interval*, it is necessary to figure out two things: how to plot one period of the signal, and then how to plot other periods. In this part, write an auxiliary function that will define the triangle wave over one period from $t = 0$ to $t = 0.04$. In other words, convert the following mathematical statement into **vectorized** MATLAB code:

$$f(t) = \begin{cases} 50t & \text{for } 0 \leq t < 0.02 \\ 2 - 50t & \text{for } 0.02 \leq t \leq 0.04 \end{cases} \tag{4}$$

There are two ways to approach this (use either one to write your triangle wave function):

[5] The function `fplot` only does scalar evaluations, but it might still be a good habit to write `fplot`'s auxiliary function to take vector inputs and produce vector outputs.

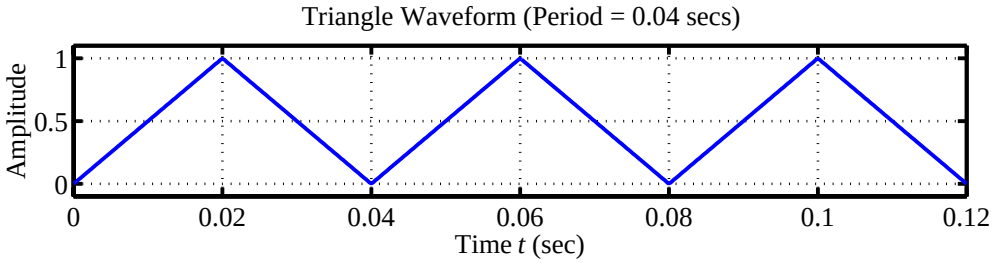



Figure 2: Triangle Wave requires a piecewise definition.

(i) Use the `find()` function to identify the subset of inputs associated with each case. For example, `jkl = find(tt>=0 & tt<=1); yy = exp(tt(jkl));` will define a signal that is $e^t$ over the interval $0 \leq t \leq 1$.

(ii) Define one period of the triangle wave by using unit-step signals to isolate different regions of the signal. For example, you can define $f(t) = e^t[u(t) - u(t-1)]$ by writing

```
function out = myfunc(tt)
out = exp(tt).*((tt>=0)-(tt>=1));
```

The logical expression `((tt>=0)-(tt>=1))` evaluates to 1 if the condition $(0 \leq t \leq 1)$ is true and it evaluates to 0 if the condition is false. Thus, multiplying by `((tt>=0)-(tt>=1))` has the effect of "switching on" the exponential function over the desired set of values of `tt`. You can use this technique to switch on and off various pieces of the function definition as required in (4). Notice that the two cases of the triangle wave define in (4) can be written as:

$$f(t) = (50t)[u(t) - u(t-0.02)] + (2 - 50t)[u(t-0.02) - u(t-0.04)] \tag{5}$$

(b) One simple way to make the function definition periodic is to use the `mod()` function which is defined in mathematics as a remainder function that gives a positive answer, e.g., $\mathrm{mod}(1.15, 0.5)$ is 0.15 because when 1.15 is divided by 0.5 the integer quotient is 2 and the remainder is 0.15. For negative numbers, the example $\mathrm{mod}(-0.7, 0.5) = 0.3$ is true because $-0.7 = 0.5(-2) + 0.3$. Consider the following vector example:

```
tt = -1:0.01:1; plot( tt, cos(pi*mod(tt,0.5)) )
```

Use this idea to define the periodic triangle wave, but put the `mod()` operation *inside* the M-file that defines the triangle wave.

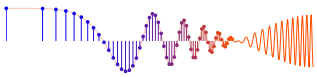
(c) Use your triangle wave function and `fplot` to make a plot of the triangle wave over the time interval $0.95 \leq t \leq 1.05$.

**Instructor Verification** (separate page)

(d) Show that you can calculate the DC value of the triangle wave by integrating over any one period of the signal. For example, you can do the integral from $t = -0.02$ to $t = +0.02$; or from $t = 0$ to $t = 0.04$. Do the integral over both ranges and compare.[6] Remember to divide by the period. Compare your answer to the correct answer which you can obtain by hand.

**Instructor Verification** (separate page)

[6] You might see warnings when `quad8` runs; these seem to be caused by the corner in the triangle wave at $t = 0.02$.



### 3.2 Using Parameters in the Integrand

One limit of the examples above is that we cannot have any variables in the function definition other than $t$. However, we want to do Fourier Series calculations, so we know that we must have the parameter $k$ and probably the period $T_0$ inside the integrals. If you look at the help on `quad8()` you will see that it has additional arguments that can handle these extra parameters. For example, we can construct a general cosine signal with different frequencies by doing the inline definition:

```
inline('cos(2*pi*f*t)','t','f')
```

Even in the case of `inline` we must be careful to identify the names of the different parameters in the function definition. The statement above tells us that $t$ and $f$ are parameters, and gives an ordering: $t$ is the first parameter and $f$ is the second one. This ordering is significant when `fplot` and `quad8` are used because they will treat the first parameter as the independent variable and all others as fixed parameters.

(a) When using `fplot` there are 3 input arguments that are usually skipped in order to pass parameters to the function definition (consult `help fplot`). Here is an example of plotting a chirp, $A\cos(\pi\alpha t^2)$ for `A=100` and `alph=13`.

```
fplot(inline('A*cos(pi*alfa*t.*t)','t','A','alfa'),[0,1],200,[],[],100,13);
```

You must have three empty matrices, or `N=200` and the two empty matrices, in the argument list, so that the number 100 will be used for the amplitude $A$, and 13 will be used for the value of $\alpha$.
Note: using `N=200` is only necessary for version 5.x, because there appears to be a bug in `fplot` in version 5.x. Picking `N=200` forces `fplot` to do a minimum number of function evaluations.

(b) When using `quad8` there are 2 input arguments that must be skipped in order to pass parameters to the function definition (consult `help quad8`). Here is an example of integrating a chirp, $A\cos(\pi\alpha t^2)$.

```
quad8(inline('A*cos(pi*alfa*t.*t)','t','A','alfa'),0,1,[],[],100,13);
```

You must have the two empty matrices in the argument list, so that the last two arguments will be used as parameters, i.e., the number 100 will be used for the amplitude $A$, and 13 will be used for the value of $\alpha$. Notice that you must use the "point-star" operator for $t^2$ because `quad8()` is expecting to evaluate the inline function for a vector of $t$'s.

(c) Follow the style of the previous two parts to make a plot of the signal:

$$x(t) = A\cos(\pi\alpha t^2)\ [u(t) - u(t-1)]$$

over the interval $-1 \le t \le 3$. For the parameters, use the values $A = 2.5$ and $\alpha = 10$. In addition, compute the definite integral over the same range.

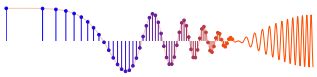
## 4 Lab Exercises: Fourier Series Coefficients

In this lab, the objective is to create a set of functions that will enable us to do the following:

1. Evaluate the Fourier Series coefficients for the following periodic signal which is defined over one period to be:

$$x(t) = 240\sin(100\pi t) \quad \text{for} \quad 0 \le t \le 1/100 \tag{6}$$

The period is $1/100$ seconds. This signal is called a full-wave *rectified* sinusoid, because it contains only the positive lobe of the sinusoidal function. Such signals and their Fourier Series are often used in DC power supply design.



2. Synthesize approximations to $x(t)$ using a finite number of Fourier Series coefficients $\{a_k\}$.

$$x_N(t) = \sum_{k=-N}^{N} a_k e^{j(2\pi/T_0)kt}$$

   where $2N+1$ is the number of terms used to form the signal and $T_0$ is the period.

3. Study and explain the convergence of the Fourier Series as $N \to \infty$.

Write down the Fourier Series integral that must be evaluated for the $x(t)$ given in Eq. (6). It is relatively easy to evaluate this integral using the techniques of calculus (required in Section 4.4), but first we are going to "do" this integral numerically using MATLAB.

## 4.1 Function for Fourier Synthesis

In this project we are going to determine Fourier series representations for periodic waveforms, synthesize the signals, and then plot them. In general, the limits on the sum in (1) are infinite, but for our computational purposes, we must restrict the limits to be a finite number $N$, which then gives the $2N+1$ term approximation:

$$x_N(t) = \sum_{k=-N}^{N} a_k e^{j(2\pi/T_0)kt}. \tag{7}$$

Sometimes it is convenient to define the *fundamental frequency* in rad/s as $\omega_0 = 2\pi/T_0$.

### 4.1.1 Fourier Synthesis: Sum of Complex Exponentials

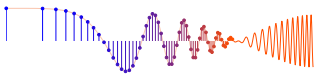
This synthesis can be done with a "sum of complex exponentials" function similar to that written for Lab #2. That M-file was called `syn_sin`. The required calling sequence of the new function, called `syn_fourier`, is given below. This MATLAB function implements the computation given in (7). When we use `syn_fourier` for Fourier synthesis, the vector of frequencies will consist of frequencies that are all integer multiplies of the fundamental frequency. In addition, we must include both the positive and negative frequency components. Therefore, the input vector of complex amplitudes `ak` will be a vector of length $L = 2N+1$ containing the $\{a_k\}$ Fourier coefficients in the order $\{a_{-N}, a_{-(N-1)}, \ldots, a_{-1}, a_0, a_1, \ldots, a_N\}$ and the vector `fk` should contain the harmonic frequencies $\{-Nf_0, -(N-1)f_0, \ldots, -f_0, 0, f_0, \ldots, Nf_0\}$ where $f_0 = 1/T_0$ is the fundamental frequency of the periodic signal.

You must write an M-file (called `syn_fourier`) that will synthesize a waveform from complex amplitude and frequency information. Since this function will be called from `fplot`, the *first* input must be the time vector. Actually, the "time" input can be a scalar because `fplot` only requests scalar evaluations from its auxiliary function (see `help fplot`). The first few statements of the M-file are the comment lines given below that explain the arguments:

```
function        xx = syn_fourier(tt, ak, fk)
%SYN_FOURIER  Function to synthesize a sum of complex
%             exponentials over the time range given by tt
%  usage:
%    xx = syn_fourier(tt, ak, fk)
%      tt = time
%           If syn_fourier is only used with fplot(), then tt can be a scalar.
%           Otherwise, tt should be a vector of times, for the time axis
%      ak = vector of complex Fourier coefficients
%      fk = vector of frequencies
```



McClellan, Schafer, and Yoder, *Signal Processing First*, ISBN 0-13-065562-7.
Prentice Hall, Upper Saddle River, NJ 07458. 



```
%             (usually contains both negative and positive freqs)
%      xx = vector of synthesized signal values. length(xx) equals length(tt)
%
%    Note: fk and ak must be the same length.
%             ak(1) corresponds to frequency fk(1),
%             ak(2) corresponds to frequency fk(2), etc.
%
% Note: the output might have a tiny imaginary part even if it
%       is supposed to be purely real.  If so, take the real part.
%
```

Several examples of Fourier synthesis of a square wave are given in the text. For 50% duty cycle square-wave case, the Fourier coefficients are given by the formula:

$$a_k = \begin{cases} \dfrac{1}{j\pi k}\left(e^{jk\pi} - 1\right) & k \neq 0 \\ 1 & k = 0 \end{cases} \tag{8}$$

This square-wave makes a convenient test case when you debug your function. For example, you can make a plot by calling the function via: `fplot('syn_fourier',[T1,T2],[],[],[],ak,fk)`.

## 4.2 Defining a Rectified Sinusoid

In this part, use the signal definition given in Eq. (6).

(a) Write the auxiliary function that will be needed to define $x(t)$ for use in `fplot()` and `quad8()`. Define the parameters carefully. Turn in the MATLAB code for this function; or if you use the inline method, give the appropriate call to `inline`.

(b) Make a plot of $x(t)$ from Eq. (6) over two periods of the signal: use the range $-1/100 \leq t \leq 1/100$. Hint: you can use MATLAB's `mod()` function to force periodicity in a definition. All you need to do is replace `t` by `mod(t,T0)` so that you always evaluate the function over the base period of definition. Here is an example that shows the periodicity of a sawtooth wave:

```
fplot('mod(t,3).*(mod(t,3)>=0 & mod(t,3)<=1)',[-5,5],200);
```

with a period of $T_0 = 3$ secs. (The argument 200 is used to force `N=200` or larger in `fplot`.)

(c) Evaluate the DC value of $x(t)$ using numerical integration with `quad8()`. Remember that this is also the $a_0$ Fourier Series coefficient.

## 4.3 Fourier Coefficients for a Full-Wave Rectified Sine Wave

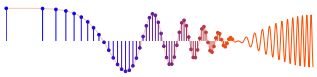
Now consider the Fourier Series analysis integral (2).

(a) In this case, the integrand is

$$x(t)e^{-j(2\pi/T_0)kt}$$

so it is necessary to define another auxiliary function and introduce another parameter, the parameter $k$. Write this auxiliary function (and turn in the code for it).

(b) Write a MATLAB function that will evaluate the Fourier Series coefficients for the rectified sine wave for $k = -N, \ldots, -1, 0, 1, 2, \ldots, N$. The function will contain a `for` loop to do all the coefficients from $k = -N$ to $k = +N$. It should return a vector containing $2N+1$ elements which are the $\{a_k\}$ coefficients. Turn in the code for this MATLAB function.



(c) Use the auxiliary function written in the previous part to evaluate the Fourier Series coefficients for $x(t)$ from Eq. (6). Find the $\{a_k\}$ coefficients for $-5 \leq k \leq 5$, and make a table of magnitude and phase versus $k$.

(d) In this part, you must synthesize a set of waveforms using different numbers of Fourier coefficients. Use `fplot` to call the `syn_fourier` synthesis function that generates $x_N(t)$ from a finite number of terms. Do the cases for $N = 2,\ N = 4$ and $N = 8$.[7] For each of these synthesized signals make a plot showing the synthesized signal $x_N(t)$ and the desired $x(t)$ on the same plot. Use a three-panel subplot to show the three cases on one page.

Note: one advantage of using `fplot` is that it will generate a time grid that is dense enough to show the details of the synthesized signal.

(e) Explain how you are getting convergence as $N$ increases. Determine the worst-case approximation error for each value of $N$ in part (d). In other words, determine *where* the error between the true signal and the Fourier approximation is the largest, and also record the size of that error.

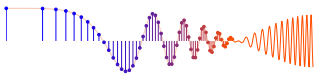
(f) How large do you have to make $N$ so that you cannot see any difference between $x(t)$ and $x_N(t)$ on a plot shown on the computer screen?

(g) Explain the empirical observations about convergence in the previous two parts by looking at the magnitudes of the $a_k$ coefficients. In other words, explain what happens when you truncate the Fourier Series sum in Eq. (1) to get the finite sum in Eq. (7). Examine the size of the terms that are omitted. Use the size of the omitted terms to justify your choice of $N$ in the previous part.

### 4.4 Mathematical Derivation

Derive the mathematical formula for the Fourier Series coefficients for $x(t)$ in Eq. (6). Verify that your formula matches the numerical computation that you did in Section 4.3(c).

[7] In order to use the `syn_fourier()` function for Fourier synthesis, we must include all the $\{a_k\}$ coefficients for both the positive and negative indices $k$. Likewise, the frequency vector `fk` must contain both positive and negative harmonics.



# Lab 13
# INSTRUCTOR VERIFICATION SHEET

*For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or professors might ask. Turn this page in at the end of your lab period.*

Name: _____ Date of Lab: _____

Part 3.1(c) Write a function that defines a periodic triangle wave, and make a plot of the signal.

Verified:_____ Date/Time:_____

Part 3.1(d) Use numerical integration to get the DC value of the periodic triangle wave. Show that the answer does not depend on which period is used in the integral. Derive (via mathematics) the numerical value of $a_0$ in the space below:

Verified:_____ Date/Time:_____



McClellan, Schafer, and Yoder, *Signal Processing First*, ISBN 0-13-065562-7.
Prentice Hall, Upper Saddle River, NJ 07458. 