A Short Tutorial on Obtaining Fourier Series Coefficients via FFT

I. Preliminaries:

1. Fourier Series:

For a given periodic function of period *P*, the Fourier series is an expansion with sinusoidal bases having periods, *P*/*n*, *n*=1, 2, … plus a constant.

Given: $f(t)$, such that $f(t+P) = f(t)$

then, with $\omega = 2\pi/P$, we expand $f(t)$ as a Fourier series by

$$\begin{aligned} f(t) = a_0 + a_1\cos(\omega t) + a_2\cos(2\omega t) + \ldots \\ + b_1\sin(\omega t) + b_1\sin(2\omega t) + \ldots \end{aligned} \qquad \text{(Eqn 1)}$$

2. Orthogonal Properties of sinusoids and cosinusoids.

It can be shown by direct integration, that the following identities are true with integers *n* and *m*:

$$\int_0^P \cos\left(\frac{2\pi}{P}nt\right)dt = 0 \qquad \text{(Eqn 2a)}$$

$$\int_0^P \sin\left(\frac{2\pi}{P}mt\right)dt = 0 \qquad \text{(Eqn 2b)}$$

$$\int_0^P \cos\left(\frac{2\pi}{P}nt\right)\cos\left(\frac{2\pi}{P}mt\right)dt = \begin{cases} 0 & \text{if } n \neq m \\ P/2 & \text{if } n = m \end{cases} \qquad \text{(Eqn 2c)}$$

$$\int_0^P \sin\left(\frac{2\pi}{P}nt\right)\sin\left(\frac{2\pi}{P}mt\right)dt = \begin{cases} 0 & \text{if } n \neq m \\ P/2 & \text{if } n = m \end{cases} \qquad \text{(Eqn 2d)}$$

$$\int_0^P \sin\left(\frac{2\pi}{P}nt\right)\cos\left(\frac{2\pi}{P}mt\right)dt = 0 \qquad \text{(Eqn 2e)}$$

Thus, applying these identities to the Fourier series expansions, we can obtain the various coefficients, i.e.,

$$a_0 = \frac{\int_0^P f(t)\,dt}{P}$$

$$a_n = \frac{\int_0^P f(t)\cos\left(n\frac{2\pi}{P}t\right)dt}{P/2} \qquad \text{(Eqn 3)}$$

$$b_n = \frac{\int_0^P f(t)\sin\left(n\frac{2\pi}{P}t\right)dt}{P/2}$$

3. Discrete Fourier Transform (DFT) :

For these transforms, we are given a time series of data, say $f(k\Delta t)$, at a uniform sampling time Δt. We will just focus here on using the computational aspects of these transforms to help us obtain the Fourier coefficients. The main reason for using DFTs is that there are very efficient methods such as Fast Fourier Transforms (FFT) to handle the numerical integration.

Given: $\hat{f}_k$, k=0,1,2,… where $\hat{f}_k = \hat{f}(k\Delta t)$

then the n^{th} DFT of $\hat{f}_k$ is defined as

$$\hat{F}_n = \sum_{k=0}^{N-1} \hat{f}_k \exp\left(-nk\frac{2\pi}{N}i\right) \qquad \text{(Eqn 4a)}$$

or after using Euler's identity,

$$\hat{F}_n = \left[\sum_{k=0}^{N-1} \hat{f}_k \cos\left(nk\frac{2\pi}{N}\right)\right] - i\left[\sum_{k=0}^{N-1} \hat{f}_k \sin\left(nk\frac{2\pi}{N}\right)\right] \qquad \text{(Eqn 4b)}$$

4. <u>Numerical Integration via Trapezoidal Rule</u>:

Let $x_k = x(k\Delta t)$, k=0,1,2,…,N where P=$N\Delta$t. Then the trapezoidal approximation of the integral is given by,

$$\begin{aligned}\int_0^P x(t)dt &\cong \left(\frac{x_0 + x_1}{2}\Delta t\right) + \left(\frac{x_1 + x_2}{2}\Delta t\right) + \cdots + \left(\frac{x_{N-1} + x_N}{2}\Delta t\right) \\ &\cong \left(\frac{x_0 + x_N}{2}\Delta t\right) + \sum_{k=1}^{N-1} x_k \Delta t \qquad \text{(Eqn 5)} \\ &\cong \Delta t \sum_{k=0}^{N-1} \hat{x}_k\end{aligned}$$

where,

$$\hat{x}_k = \begin{cases} \left(\frac{x_0 + x_N}{2}\right) & \text{if} \quad k = 0 \\ x_k & \text{if} \quad k = 1,2,\ldots,N-1 \end{cases}$$

II. Main Procedure:

Step 1. Given the function, $f(t)$, $0 \le t \le P$, discretize the time span such that N is a power of 2, i.e. $N = 2^m$, for some integer m. (This is to take advantage of the efficiency of the FFT algorithm).

Thus, $\Delta t = 2^{-m} P$.

Step 2. Recast the raw data as

$$\hat{f}_k = \begin{cases} \left(\frac{f_0 + f_N}{2}\right) & \text{if} \quad k = 0 \\ f_k & \text{if} \quad k = 1,2,\ldots,N-1 \end{cases} \qquad \text{(Eqn 6)}$$

Step 3. Obtain the FFT of $\hat{f}_k$ (via Matlab®, MathCad® or Excel®)

$$\mathbf{F} = \mathbf{FFT}(\hat{f})$$

where, **F**, is a vector of discrete Fourier transforms whose n^{th} element is the $(n\text{-}1)^{th}$ transform (since most vectors start with index 1 instead of index 0, with the exception of MathCad®), i.e.

$$\mathbf{F}_{n+1} = \sum_{k=0}^{N-1} \hat{f}_k \exp\left(-nk\frac{2\pi}{N}i\right), \qquad n = 0,1,\ldots \qquad \text{(Eqn 7)}$$

Step 4. Since $P=N\Delta t$ and $t_k=k\Delta t$, when applying trapezoidal rule (5) into (3), while using the computational results of (7), we have

$$\begin{aligned} a_0 &= \frac{1}{N}\mathrm{Re}(\mathbf{F}_1) \\ a_n &= \frac{2}{N}\mathrm{Re}(\mathbf{F}_{n+1}) \\ b_n &= \frac{2}{N}\mathrm{Im}(\mathbf{F}_{n+1}) \end{aligned} \qquad \text{(Eqn 8)}$$

Note: Because of the "butterfly" algorithm of FFT, only the first half of the discrete transform can be used, while the other half is simply the reflection/conjugate of the first half. This means using this method, we can only obtain a_n and b_n, with n = 0,1,2,…,N/2. When N is large, this does not usually present a serious limitation. For instance, with $N=2^{10}$, $N/2=512$.

III. Example 1.

Suppose we have the following function that is a finite sum of sine and cosine functions:

$$f(t) = 5 - 2\cos(\omega t) + 3\cos(5\omega t) + 8\sin(20\omega t) \quad ; \quad \omega = \frac{2\pi}{10}$$

The plot is shown in Figure 1. The fundamental period is 10 sec.

A sample m-file in Matlab could be run similar to the one shown in Figure 2.

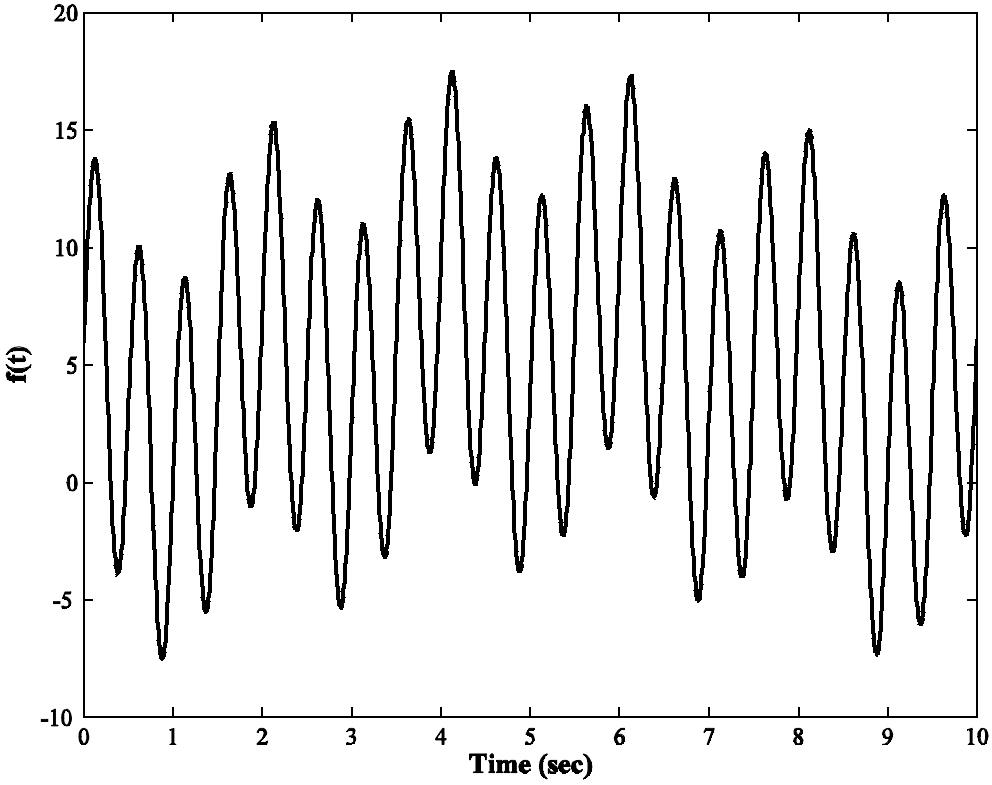

Figure 1.

```
%
% initializations
% ===============

    T   = 10    ;   % period (secs/cycle)
    m   = 11    ;   %
    N   = (2^m) ;   % number of discrete data

    ffreq = 2*pi/T ;    % fundamental frequency

    t   = linspace(0,T,N+1) ;

    f   = 5 -2*cos(    ffreq*t )...
            +3*cos(  5*ffreq*t )...
            +8*sin( 20*ffreq*t );

    fhat        = f                 ;
    fhat(1)     = (f(1)+f(N+1))/2   ;
    fhat(N+1)   = []                ;

    F           = fft(fhat,N)       ;

%
% use only the first half of the DFTs
% ===================================
%

    F=F(1:N/2)      ;
    k=0:(N/2-1)     ;
    omega=k*ffreq   ; % in units of rads/sec

    % extracting the coefficients
    % ---------------------------

    A   = 2*real(F)/N         ;
    A(1)= A(1)/2              ;
    B   =-2*imag(F)/N         ;
```

Figure 2.

Thus, after these commands are implemented, we will obtain exactly the coefficients we started with, i.e: a_0=A(1)=5, a_1=A(2)=-2, a_5=A(6)=3, while a_k=0, $k \neq 0,1, 5$. Also, we find b_{20} =B(21)= 8, while b_k=0, $k \neq 20$.

This example shows that using FFT, we can recover the coefficients exactly. However, it must be noted that we had used a periodic function and we had discretized our time domain to be exactly 2^m. This is usually not the case when applying the procedure. Nonetheless, we can show that the procedure will still obtain a reasonable approximation as we show in the next example.

IV. Example 2.

Consider the following continuous, piecewise-smooth and non-periodic function:

$$f(t) = \begin{cases} 5 & \text{if } t < 20 \\ t/4 \cos\left(\frac{2\pi}{20} t\right) & \text{if } 20 \le t < 80 \\ 28 - t/10 & \text{if } t \ge 80 \end{cases}$$

the plot of f(t) is shown in Figure 3.

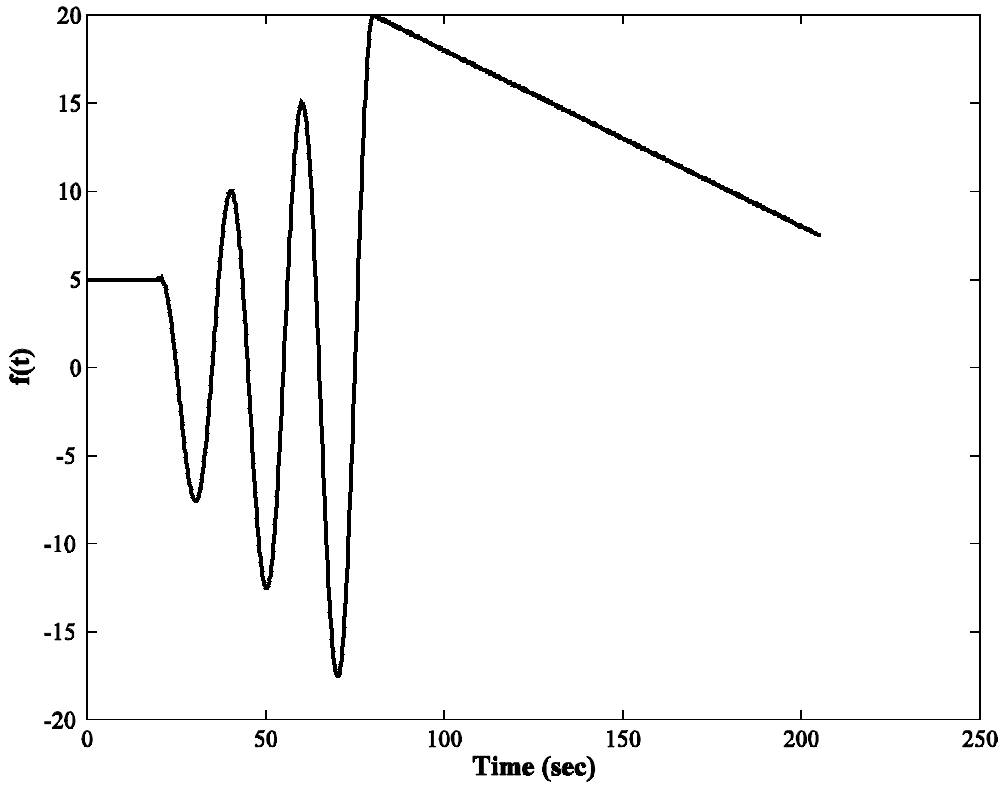

Figure 3.

As in example 1, we could run an m-file in Matlab® similar to that shown in Figure 4.

```
%
% initializations
% ===============

    m   = 11    ;   %
    N   = (2^m) ;   % number of discrete data
    Dt  = 0.1   ;
    T   = N*Dt  ;

    ffreq = 2*pi/T ;    % fundamental frequency

    t   = linspace(0,T,N+1) ;
    f   = [];

    for tk=t

        if tk<20
            x   = 5                 ;
            f   = [f,x]             ;
        elseif tk<80
            x   = tk/4*cos(2*pi*tk/20);
            f   = [f,x]             ;
        else
            x   = -tk/10+28         ;
            f   = [f,x]             ;
        end
    end


    fhat        = f                 ;
    fhat(1)     = (f(1)+f(N+1))/2   ;
    fhat(N+1)   = []                ;

    F           = fft(fhat,N)       ;

%
% use only the first half of the DFTs
% ==================================
%

    F=F(1:N/2)      ;
    k=0:(N/2-1)     ;
    omega=k*ffreq   ; % in units of rads/sec

    % extracting the coefficients
    % ---------------------------

    A   = 2*real(F)/N     ;
    A(1)= A(1)/2          ;
    B   =-2*imag(F)/N     ;
```

Figure 4.

Having found the vector A and B, which contains the values of a_0, a_1, a_2,…, b_1, b_2,… needed for the Fourier series, we could truncate the number of terms to obtain the desired accuracy, i.e. via an m-file similar to that shown in Figure 5, the value of `L` will mean the number of terms used for the approximation.

```
L   = 10                          ;
fapprox = A(1)*ones(size(t))      ;
for k=1:L
    fapprox = fapprox + A(k+1)*cos(omega(k+1)*t)...
            + B(k+1)*sin(omega(k+1)*t);
end
```

Figure 5.

So for `L`=10 (see Figure 5), `L`=15 (see Figure 6) and `L`=100 (see Figure 7), we see that the approximation gets better with the inclusion of more terms.

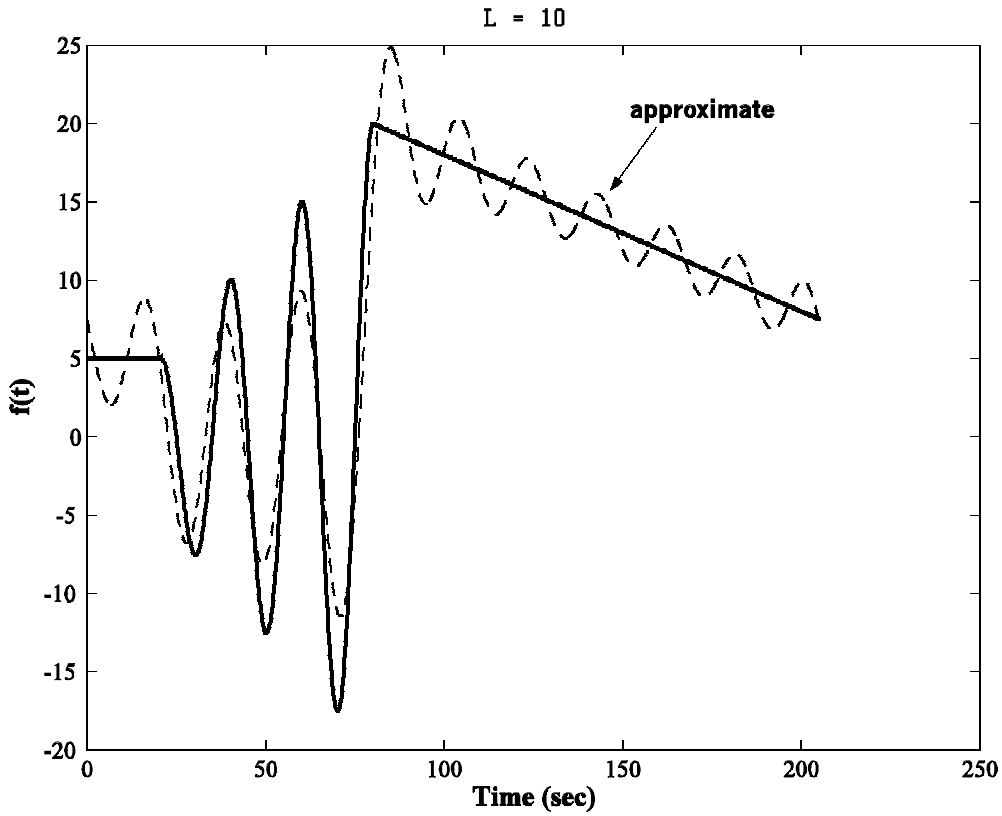

Figure 6.

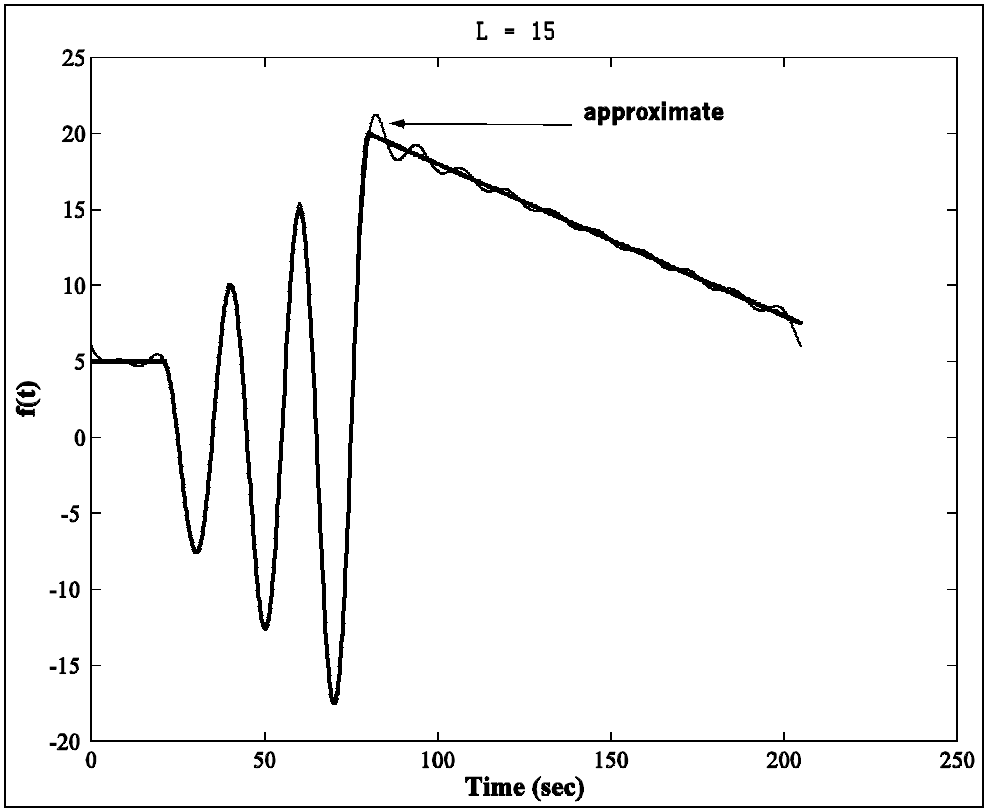

Figure 7

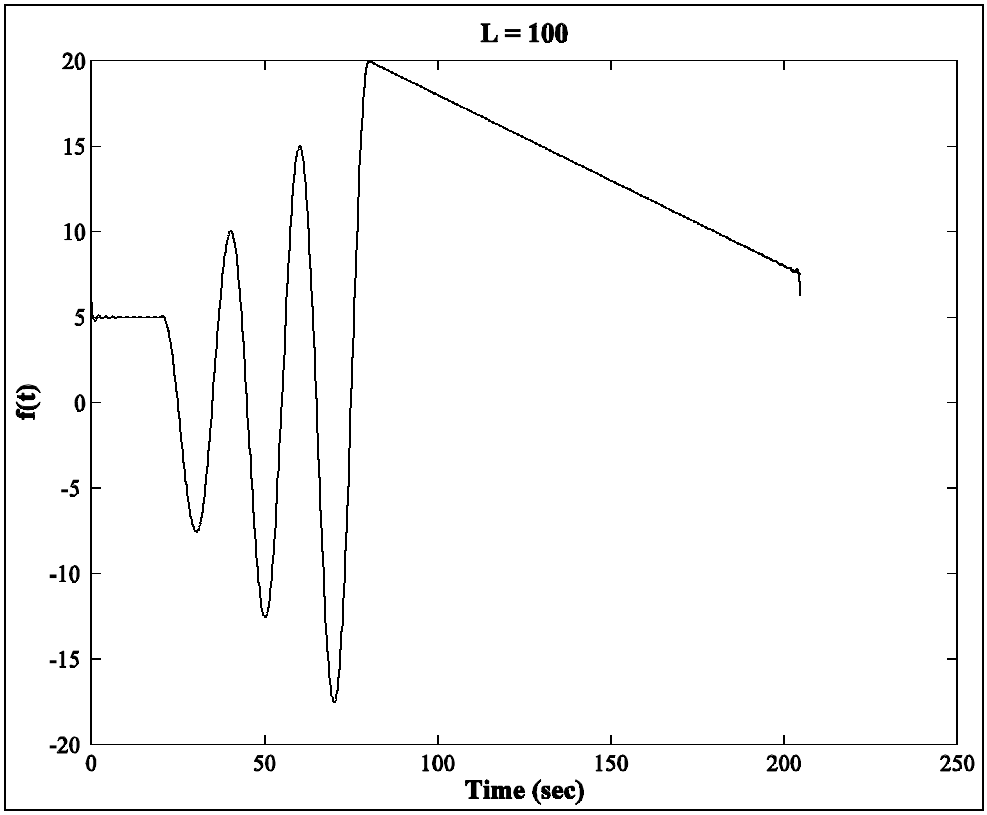

Figure 8

Another way to decide how many terms to include in the approximation is to plot the magnitude of the Fourier coefficients, e.g.

```
>> power=sqrt(A.^2 + B.^2);
>> plot(power)
>> axis([0 50 0 10])
```

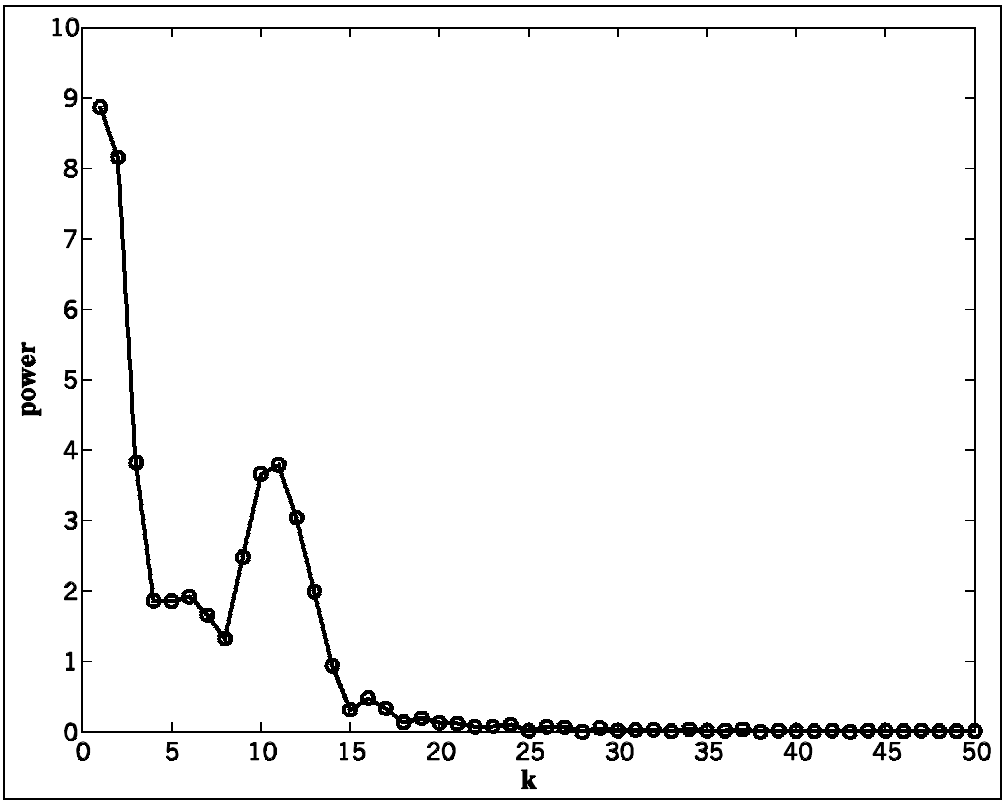

Figure 9.

From Figure 9, we see that around $k>30$, the contribution of these terms is minimal to improving the approximation. This means the plot shown in Figure 8 contains unnecessarily large amount of Fourier terms.