Heduna publications of International Research and Reviews

Journal homepage: www. hedunapublications.com

ISBN - 978-81-963578-0-1

"Analysis of Fourier series using Python Code"

Shyamal Bhar
Assistant Professor
Department of Physics
Vidyasagar College for Women, Kolkata, West Bengal, India
shyamal.bhar@gmail.com

Abstract

Researchers now focus on computation and simulation in various disciplines. Different problems in Physics are now being solved numerically with the help of a powerful programming language. Python is a modern, flexible, and intelligent programming language with many scientific capabilities for solving and understanding Physics problems. In this article we shall apply python programming language to analyze Fourier series of different functions.

Keywords

Fourier series, Dirichlet conditions, python, square wave, triangular wave, sawtooth wave

Introduction

Now-a-days computation and simulation in different fields has attracted more attention to the researchers. Many physical problems can be easily solved numerically. To solve and understand any physical problem numerically we need to use some powerful programming language. Python is a very versatile programming language that can be used for both scientific and computational physics. It is a well designed modern programming language that is easy to learn and implement without having vast experience in past programming. Python is used in Physics as a tool for data analysis. Python has many powerful scientific modules such as numpy, scipy, pandas etc. It has also a plotting module such as matplotlib for the graphical representation of scientific problems. All these modules have various powerful functions that can be used in solving and understanding many mathematical and physical problems. Python is widely used in physics as it has many modules which provide useful functions for Physics calculations. In this article we shall use python programming language to analyze Fourier series

of different functions. Expressing a function in Fourier series plays an important role both in Physics and Mathematics.

Brief Introduction to Fourier series

We know that there are many ways by which any complicated function may be expressed as power series. This is not the only way in which a function may be expressed as a series but there is a method of expressing a periodic function as an infinite sum of sine and cosine functions. This representation is known as Fourier series. The computation and study of Fourier series is known as harmonic analysis and is useful as a way to break up an arbitrary periodic function into a set of simple harmonic terms that can be plugged in, solved individually, and then recombined to obtain the solution to the original problem or an approximation to it to whatever accuracy is desired. Unlike Taylor series, a Fourier series can describe functions that are not everywhere continuous and/or differentiable. There are other advantages of using trigonometric terms. They are easy to differentiate and integrate and each term contain only one characteristic frequency. Analysis of Fourier series becomes important because this method is used to represent the response of a system to a periodic input and the response depends on the frequency content of the input.

- **Dirichlet Conditions**

The conditions that a function $f(x)$ may be expressed as Fourier series are known as the Dirichlet conditions. The conditions are

i) The function must be periodic.
ii) It must be single valued and continuous. There may a finite number of finite discontinuities.
iii) It must have only a finite number of maxima and minima within one period.
iv) The integral over one period of $|f(x)|$ must converge.

- **Orthogonality Relations among Fourier Components**

Fourier series makes of the orthogonality relationships of the sine and cosine functions. The integral over one period of the product of any two terms have the following properties:

$$\int_{x_0}^{x_0+L} \sin\left(\frac{2\pi nx}{L}\right)\cos\left(\frac{2\pi mx}{L}\right)dx = 0 \text{ for all } m \text{ and } n$$

$$\int_{x_0}^{x_0+L} \cos\left(\frac{2\pi nx}{L}\right)\cos\left(\frac{2\pi mx}{L}\right)dx = \begin{cases} L \text{ for } m = n = 0 \\ \frac{L}{2} \text{ for } m = n > 0 \\ 0 \text{ for } m \neq n \end{cases}$$

$$\int_{x_0}^{x_0+L} \sin\left(\frac{2\pi nx}{L}\right)\sin\left(\frac{2\pi mx}{L}\right)dx = \begin{cases} 0 \text{ for } m=n=0 \\ \frac{L}{2} \text{ for } m=n>0 \\ 0 \text{ for } m\neq n \end{cases}$$

- **Expansion of a function in Fourier series**

So the Fourier series of the function $f(x)$ over the periodic interval $[0,L]$ is written as

$$f(x') = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_n \cos\left(\frac{2\pi nx'}{L}\right) + b_n \sin\left(\frac{2\pi nx'}{L}\right)\right]$$

where a_n and b_n are constants called the Fourier coefficients and the coefficients are expressed as

$$a_0 = \frac{2}{L}\int_0^L f(x')\,dx'$$

$$a_n = \frac{2}{L}\int_0^L f(x')\cos\left(\frac{2\pi nx'}{L}\right)dx'$$

$$b_n = \frac{2}{L}\int_0^L f(x')\sin\left(\frac{2\pi nx'}{L}\right)dx'$$

Determination of these coefficients is required for the expansion of a function in Fourier series.

- **Expansion of a function in Fourier series with other intervals**

i) The Fourier series of the function $f(x)$ over the periodic interval $[-L,L]$ is written as

$$f(x') = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_n \cos\left(\frac{\pi nx'}{L}\right) + b_n \sin\left(\frac{\pi nx'}{L}\right)\right]$$

where,

$$a_0 = \frac{1}{L}\int_{-L}^{L} f(x')\,dx'$$

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x')\cos\left(\frac{\pi nx'}{L}\right)dx'$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x')\sin\left(\frac{\pi nx'}{L}\right)dx'$$

ii) The Fourier series of the function $f(x)$ over the periodic interval $[-\pi,\pi]$ is written as

$$f(x') = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_n \cos(nx') + b_n \sin(nx')\right]$$

where,

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x')\,dx'$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x')\cos(nx')\,dx'$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x')\sin(nx')\,dx'$$

Generation of different piecewise continuous functions from python libraries

We need to construct piecewise continuous functions for expressing these functions into infinite sum of sine and cosine functions. Here we shall produce signal waveforms of various types, shapes, frequencies. There are many waveforms available in the **scipy module** of python. We shall also generate user defined functions for the purpose of analysis using Fourier series.

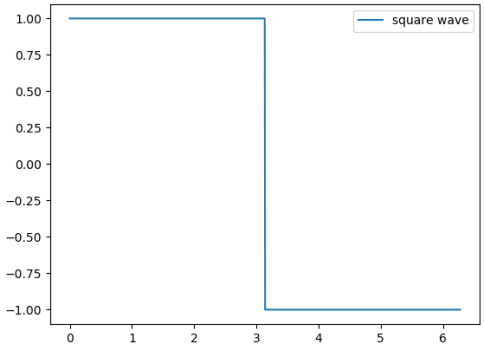
1. Square Wave

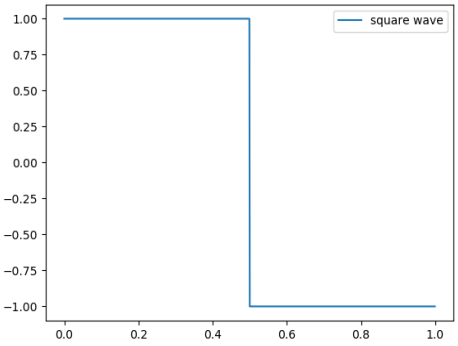
A square wave is a non-sinusoidal periodic waveform in which the amplitudes alternate between maximum and minimum value with equal duration. The pulse duration of the positive amplitude must be equal to the pulse duration of the negative amplitude or zero. It is a special case of a pulse wave in which pulse duration between maximum and minimum is arbitrary. The ratio of pulse duration for maximum amplitude to the total pulse duration is called duty cycle. For square wave the duty cycle is 0.5.

The square wave which contains only the odd harmonics is generated from scipy.signal.square module and its general syntax is given below.

Scipy.signal.square (x, duty=0.5)

This returns a periodic square-wave waveform. The square wave has a period 2π and has value +1 from 0 to $2\pi * duty$ and -1 from $2\pi * duty$ to 2π. The argument *duty* must be in the interval [0, 1].



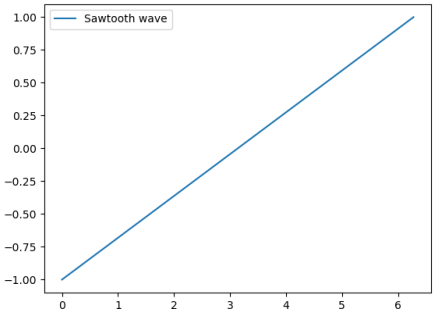


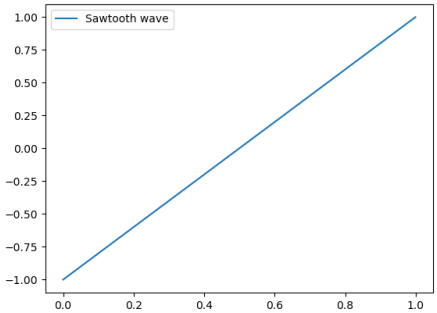
2. The sawtooth wave

The sawtooth waveform is also a non-sinusoidal waveform looks like a teeth of a plain-tooth saw. It consists of a horizontal and vertical deflection signals. The sawtooth waveform is generated from the scipy.signal.sawtooth module.

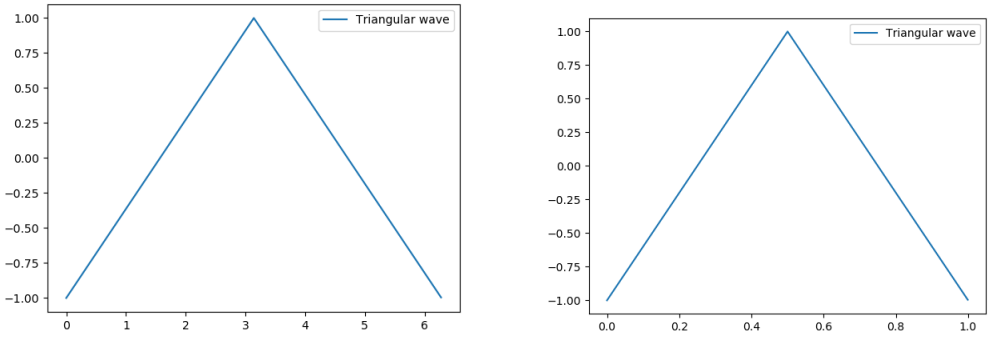
Scipy.signal.sawtooth(x, width=1)

It returns a periodic sawtooth or triangle waveform.
The sawtooth waveform has a period 2π, rises from -1 to 1 on the interval 0 to $2\pi * width$, then drops from 1 to -1 on the interval $2\pi * width$ to 2π. The *width* must be in the interval [0, 1].





Sawtooth wave form with different periods



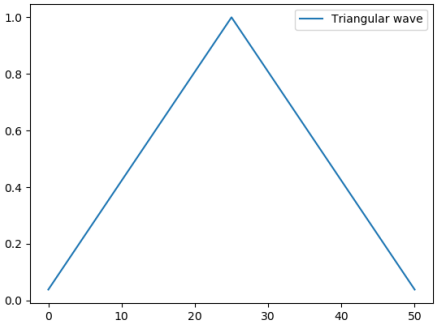
Triangular waveform from sawtooth waveform with different periods

3. Triangular Waveform

Triangular wave is a non-sinusoidal wave of triangle shaped. Triangular wave contains only odd harmonics. The general syntax for generating triangular wave form scipy module is discussed below.

scipy.signal.triang(M, sym=True)

This returns a triangular window. Here M is the number of points in the output window. The window (w) has a maximum value normalized to 1 (though the value 1 does not appear if M is even and *sym* is True).

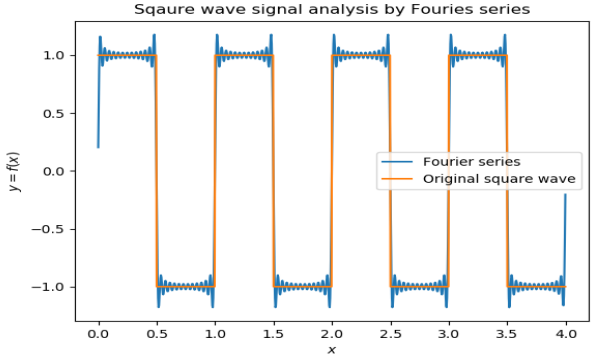


Triangular wave form from triang() function

Fourier series analysis of different functions using python

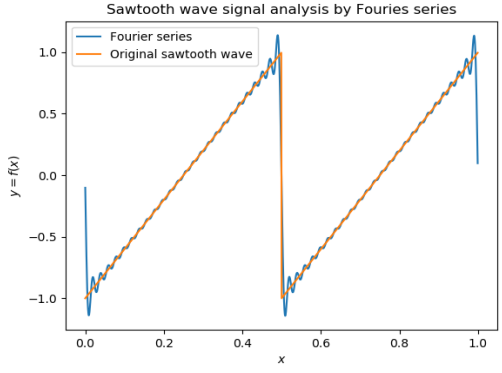
1. **Fourier series analysis of a square wave of the interval** $[0,\ L]$

```
# Fourier series analysis for a square wave function

# Importing python libraries
import numpy as np
from scipy.signal import square
import matplotlib.pyplot as plt
from scipy.integrate import simps

L=4             # Periodicity of the periodic function f(x)
freq=4          # No of waves in time period L
dutycycle=0.5
samples=1000
terms=100

# Generation of square wave
x=np.linspace(0,L,samples,endpoint=False)
y=square(2.0*np.pi*x*freq/L,duty=dutycycle)

# Calculation of Fourier coefficients
a0=2./L*simps(y,x)
an=lambda n:2.0/L*simps(y*np.cos(2.*np.pi*n*x/L),x)
bn=lambda n:2.0/L*simps(y*np.sin(2.*np.pi*n*x/L),x)

# sum of the series
s=a0/2.+sum([an(k)*np.cos(2.*np.pi*k*x/L)+bn(k)*np.sin(2.*np.pi*k*x/L) for k in
range(1,terms+1)])

# Plotting
plt.plot(x,s,label="Fourier series")
plt.plot(x,y,label="Original square wave")
plt.xlabel("$x$")
plt.ylabel("$y=f(x)$")
plt.legend(loc='best',prop={'size':10})
plt.title("Sqaure wave signal analysis by Fouries series")
plt.savefig("fs_square.png")
plt.show()
```



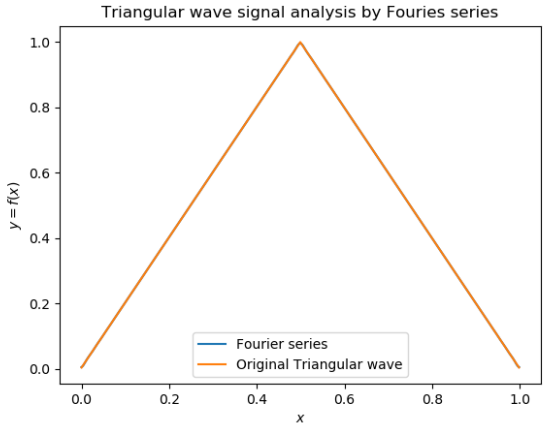
2. **Fourier series analysis of a sawtooth wave of the interval $[0, L]$**

```
# Fourier series analysis for a sawtooth wave

# Importing python libraries
import numpy as np
from scipy.signal import square,sawtooth
import matplotlib.pyplot as plt
from scipy.integrate import simps

L=1             # Periodicity of the periodic function f(x)
freq=2          # No of waves in time period L
width_range=1
samples=1000
terms=50

# Generation of Sawtooth function

x=np.linspace(0,L,samples,endpoint=False)
y=sawtooth(2.0*np.pi*x*freq/L,width=width_range)

# Calculation of Co-efficients
a0=2./L*simps(y,x)
an=lambda n:2.0/L*simps(y*np.cos(2.*np.pi*n*x/L),x)
bn=lambda n:2.0/L*simps(y*np.sin(2.*np.pi*n*x/L),x)

# Sum of the series
s=a0/2.+sum([an(k)*np.cos(2.*np.pi*k*x/L)+bn(k)*np.sin(2.*np.pi*k*x/L) for k in
range(1,terms+1)])

# Plotting

plt.plot(x,s,label="Fourier series")
plt.plot(x,y,label="Original sawtooth wave")
plt.xlabel("$x$")
plt.ylabel("$y=f(x)$")
plt.legend(loc='best',prop={'size':10})
plt.title("Sawtooth wave signal analysis by Fouries series")
plt.savefig("fs_sawtooth.png")
plt.show()
```



3. Fourier series analysis of a triangular wave form of the interval $[0,\ L]$

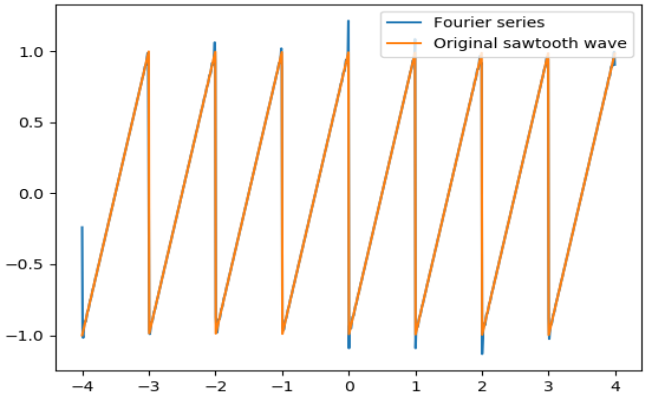
```
# Fourier series analysis for a Triangular wave function

# Importing python libraries
import numpy as np
from scipy.signal import square,sawtooth,triang
import matplotlib.pyplot as plt
from scipy.integrate import simps


L=1          # Periodicity of the periodic function f(x)
samples=501
terms=50

# Generation of Triangular wave
x=np.linspace(0,L,samples,endpoint=False)
y=triang(samples)

# Fourier Coefficients
a0=2./L*simps(y,x)
an=lambda n:2.0/L*simps(y*np.cos(2.*np.pi*n*x/L),x)
bn=lambda n:2.0/L*simps(y*np.sin(2.*np.pi*n*x/L),x)

# Series sum
s=a0/2.+sum([an(k)*np.cos(2.*np.pi*k*x/L)+bn(k)*np.sin(2.*np.pi*k*x/L) for k in
range(1,terms+1)])

# Plotting
plt.plot(x,s,label="Fourier series")
plt.plot(x,y,label="Original Triangular wave")
plt.xlabel("$x$")
plt.ylabel("$y=f(x)$")
plt.legend(loc='best',prop={'size':10})
plt.title("Triangular wave signal analysis by Fouries series")
plt.savefig("fs_triangular.png")
plt.show()
```



4. **Fourier series analysis of a sawtooth wave of the interval $[-L,\ L]$.**

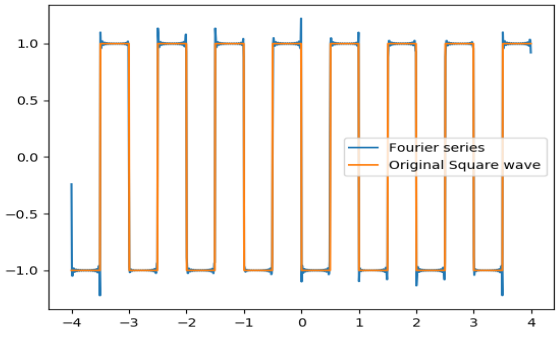
This waveform is not generated from scipy module rather it is user defined sawtooth wave.

```
# Fourier series analysis for a sawtooth wave function
# User defined function

# Importing python libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simps


L=1.0 # half wavelength, Wavelength=2L
freq=2  #  frequency
samples=1001
terms=300

# Defining sawtooth function
x=np.linspace(-L,L,samples,endpoint=False)
f=lambda x: (freq*x%(2*L)-L)/L

# Fouriers coefficients

a0=1./L*simps(f(x),x)
an=lambda n:1.0/L*simps(f(x)*np.cos(1.*np.pi*n*x/L),x)
bn=lambda n:1.0/L*simps(f(x)*np.sin(1.*np.pi*n*x/L),x)

# Series sum
xp=4*x
s=a0/2.+sum([an(k)*np.cos(1.*np.pi*k*xp/L)+bn(k)*np.sin(1.*np.pi*k*xp/L) for k in
range(1,terms+1)])

# Plotting
plt.plot(xp,s,label="Fourier series")
plt.plot(xp,f(xp),label="Original sawtooth wave")
plt.legend(loc='best',prop={'size':10})
plt.savefig("saw_ud.png")
plt.show()
```



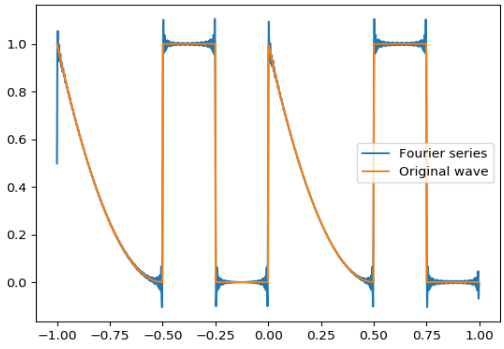
5. **Fourier series analysis of a square wave of the interval $[-L,\ L]$.**
   This waveform is not generated from scipy module rather it is user defined square wave.

```
# Fourier series analysis for a square wave function
# User defined function

# Importing python libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simps

L=1.0 # half wavelength, Wavelength=2L
freq=2  #  frequency
samples=1001
terms=300
# Generating Square wave
x=np.linspace(-L,L,samples,endpoint=False)
F=lambda x: np.array([-1 if -L<=u<0 else 1 for u in x])
f=lambda x: F(freq*x%(2*L)-L)

# Fourier Coefficients
a0=1./L*simps(f(x),x)
an=lambda n:1.0/L*simps(f(x)*np.cos(1.*np.pi*n*x/L),x)
bn=lambda n:1.0/L*simps(f(x)*np.sin(1.*np.pi*n*x/L),x)

# Series sum
xp=4*x
s=a0/2.+sum([an(k)*np.cos(1.*np.pi*k*xp/L)+bn(k)*np.sin(1.*np.pi*k*xp/L) for k in
range(1,terms+1)])

#Plotting
plt.plot(xp,s,label="Fourier series")
plt.plot(xp,f(xp),label="Original
Square wave")
plt.legend(loc='best',prop={'size':10})
plt.savefig("square_ud.png")
plt.show()
```



6. Fourier series analysis of an arbitrary function of the interval $[-L,\ L]$.

```
# Fourier series analysis for a Arbitrary waves function
# User defined function
# Importing python libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simps

L=1.0 # half wavelength, Wavelength=2L
freq=2  #  frequency

samples=1001
terms=300
# Generating  wave
x=np.linspace(-L,L,samples,endpoint=False)
F=lambda x: np.array([u**2 if -L<=u<0 else 1 if 0<u<0.5 else 0 for u in x])
#F=lambda x: abs(np.sin(2*np.pi*x))
f=lambda x: F(freq*x%(2*L)-L)

# Fourier Coefficients
a0=1./L*simps(f(x),x)
an=lambda n:1.0/L*simps(f(x)*np.cos(1.*np.pi*n*x/L),x)
bn=lambda n:1.0/L*simps(f(x)*np.sin(1.*np.pi*n*x/L),x)

# Series sum
xp=x
s=a0/2.+sum([an(k)*np.cos(1.*np.pi*k*xp/L)+bn(k)*np.sin(1.*np.pi*k*xp/L) for k in
range(1,terms+1)])
#Plotting
plt.plot(xp,s,label="Fourier series")
plt.plot(xp,f(xp),label="Original wave")
plt.legend(loc='best',prop={'size':10})
plt.savefig("arb_ud.png")
plt.show()
```



References

1. Mathematical Methods for Physicists by Arfken; Elsevier Publisher
2. Mathematical methods of Physics and Engineering: A comprehensive guide by K.F. Riley, M.P. Hobson, S.J. Bence; Cambridge University Press
3. Higher Engineering Mathematics by B. S. Grewal; Khanna Publishers
4. https://www.gogreenva.org/advantages-of-using-python-to-teach-physics/
5. Computational Methods for Physics, Joel Franklin; Cambridge University Press
6. Programming for Computation Python, Svein Linge, Hans Petter Lantangen; Springer
7. Numerical Python, Robert Johansson; Apress Publication
8. Scientific Computing in python, Avijit Kar Gupta, Techno World.