FOURIER SERIES

Pawanpreet Kaur (2020PHY1092) (20068567038)

Monu Chaurasiya (2020PHY1102) (20068567035)

Prateek Bhardwaj (2020PHY1110) (20068567042)

S.G.T.B. Khalsa College, University of Delhi, Delhi-110007, India.

Practical Report Submitted to

Dr. Mamta

"32221401 - Mathematical Physics III"

Table of Contents

1 Theory

1.1 Dirichlet Conditions

The Dirichlet conditions are sufficient conditions for a real-valued, periodic function f to be equal to the sum of its Fourier series at each point where f is continuous.
The conditions are:

- $f(x)$ should be single valued and bounded.
- There should be finite number of minima or maxima in given interval.
- $f(x)$ should be piecewise continuous with finite number of discontinuity in the given interval but the amount of discontinuity should be finite

1.2 Fourier Representation of a Periodic Function

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n cos\left(\frac{2n\pi x}{b-a}\right) + \sum_{n=1}^{\infty} b_n sin\left(\frac{2n\pi x}{b-a}\right)$$

where,

$$a_0 = \frac{2}{b-a}\int_a^b f(x)dx$$

$$a_n = \frac{2}{b-a}\int_a^b f(x)cos\left(\frac{2n\pi x}{b-a}\right)dx \qquad n = 1,2....$$

$$b_n = \frac{2}{b-a}\int_a^b f(x)sin\left(\frac{2n\pi x}{b-a}\right)dx \qquad n = 1,2....$$

1.2.1 Derivation of Expressions for Fourier Coefficients

(i) a_0

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

Taking L=π,

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

$$\int_{-\pi}^{\pi} f(x)dx = \int_{-\pi}^{\pi} \frac{a_0}{2}dx + \sum_{n=1}^{\infty}\int_{-\pi}^{\pi}[a_n cos(nx)]dx + \sum_{n=1}^{\infty}\int_{-\pi}^{\pi}[b_n sin(nx)]dx$$

$$\int_{-\pi}^{\pi} f(x)dx = \frac{a_0}{2}[x]_{-\pi}^{\pi} + 0 + 0$$

$$\int_{-\pi}^{\pi} f(x)dx = \frac{a_0}{2}[\pi - (-\pi)]$$

$$\int_{-\pi}^{\pi} f(x)dx = \pi a_0$$

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

or

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

(ii) $\mathbf{a}_n$

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

Taking L=π,

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

$$\int_{-\pi}^{\pi} f(x)cos(mx)dx = \frac{a_0}{2}\int_{-\pi}^{\pi} cos(mx)dx + \sum_{n=1}^{\infty} a_n \int_{-\pi}^{\pi}[cos(nx)cos(mx)]dx + \sum_{n=1}^{\infty} b_n \int_{-\pi}^{\pi}[sin(nx)cos(mx)]dx$$

Taking m=n and using standard integrals,

$$\int_{-\pi}^{\pi} f(x)cos(nx)dx = \frac{a_0}{2}[\frac{sinnx}{n}]_{-\pi}^{\pi} + \sum_{n=1}^{\infty} a_n\pi + \sum_{n=1}^{\infty} b_n(0)$$

$$\int_{-\pi}^{\pi} f(x)cos(nx)dx = 0 + a_n\pi + 0 \qquad n = 1,2,3.....\infty$$

$$\int_{-\pi}^{\pi} f(x)cos(nx)dx = a_n\pi \qquad n = 1,2,3.....\infty$$

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

or

$$\boxed{a_n = \frac{1}{L}\int_{-L}^{L} f(x)cos(\frac{n\pi x}{L})dx}$$

(iii) $\mathbf{b}_n$

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

Taking L=π,

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

$$\int_{-\pi}^{\pi} f(x)sin(mx)dx = \frac{a_0}{2}\int_{-\pi}^{\pi} sin(mx)dx + \sum_{n=1}^{\infty} a_n \int_{-\pi}^{\pi}[cos(nx)sin(mx)]dx + \sum_{n=1}^{\infty} b_n \int_{-\pi}^{\pi}[sin(nx)sin(mx)]dx$$

Taking m=n and using standard integrals,

$$\int_{-\pi}^{\pi} f(x)sin(nx)dx = \frac{a_0}{2}[\frac{-cosnx}{n}]_{-\pi}^{\pi} + \sum_{n=1}^{\infty} a_n(0) + \sum_{n=1}^{\infty} b_n\pi$$

$$\int_{-\pi}^{\pi} f(x)sin(nx)dx = \frac{a_0}{2}(0) + 0 + b_n\pi \qquad n = 1,2,3.....\infty$$

$$\int_{-\pi}^{\pi} f(x)sin(nx)dx = b_n\pi \qquad n = 1,2,3.....\infty$$

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

or

$$\boxed{b_n = \frac{1}{L}\int_{-L}^{L} f(x)sin(\frac{n\pi x}{L})dx}$$

1.2.2 If the function f(x) is an even or odd function

For even functions:

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

$$a_n = \frac{2}{L}\int_0^L f(x)cos(\frac{n\pi x}{L})dx$$

$$b_n = 0$$

For odd functions:

$$a_0 = 0$$

$$a_n = 0$$

$$b_n = \frac{2}{L}\int_0^L f(x)sin(\frac{n\pi x}{L})dx$$

1.2.3 Fourier Series Representation

(i)

$$f(x) = \begin{cases} 0 & \text{if} \quad -1 < x < 0 \\ 1 & \text{if} \quad 0 < x < 1 \end{cases}$$
$$f(x+2) = f(x)$$

<u>Solution:</u>

$$b - a = 2$$

We know that,

$$a_o = \frac{2}{b-a}\int_a^b f(x)\,dx$$
$$a_n = \frac{2}{b-a}\int_a^b f(x)cos\left(\frac{2n\pi x}{b-a}\right)dx$$
$$b_n = \frac{2}{b-a}\int_a^b f(x)sin\left(\frac{2n\pi x}{b-a}\right)dx$$

$$a_o = \frac{2}{2}\int_{-1}^1 f(x)\,dx$$
$$= \int_{-1}^0 (0)dx + \int_0^1 (1)dx$$
$$= 0 + [x]_0^1$$
$$a_o = 1$$

$$a_n = \frac{2}{2}\int_{-1}^1 f(x)cos\left(\frac{2n\pi x}{2}\right)dx$$
$$= \int_{-1}^1 f(x)cos\,(n\pi x)\,dx$$
$$= \int_{-1}^0 (0)cos(n\pi x)dx + \int_0^1 (1)cos(n\pi x)dx$$
$$= \left[\frac{sin(n\pi x)}{n\pi}\right]_0^1$$
$$= \frac{sin(n\pi) - sin(0)}{n\pi}$$
$$a_n = 0$$

$$b_n = \frac{2}{2}\int_{-1}^1 f(x)sin\left(\frac{2n\pi x}{2}\right)dx$$
$$= \int_{-1}^1 f(x)sin\,(n\pi x)\,dx$$

$$= \int_{-1}^{0} (0) sin(n\pi x) dx + \int_{0}^{1} (1) sin(n\pi x) dx$$

$$= \left[\frac{-cos(n\pi x)}{n\pi} \right]_0^1$$

$$= \frac{1 - cos(n\pi)}{n\pi}$$

$$b_n = \frac{1 - (-1)^n}{n\pi}$$

$$f(x) = \frac{1}{2} + \sum_{n=1}^{\infty} \left(\frac{1 - (-1)^n}{n\pi} \right) sin(n\pi x)$$

(ii)

$$f(x) = \begin{cases} 0 & \text{if} \quad -1 < x < -0.5 \\ 1 & \text{if} \quad -0.5 < x < 0.5 \\ 0 & \text{if} \quad 0.5 < x < 1 \end{cases}$$

$$f(x+2) = f(x)$$

<u>Solution:</u>

$$b - a = 2$$

We know that,

$$a_o = \frac{2}{b-a} \int_a^b f(x)\, dx$$

$$a_n = \frac{2}{b-a} \int_a^b f(x) cos \left(\frac{2n\pi x}{b-a} \right) dx$$

$$b_n = \frac{2}{b-a} \int_a^b f(x) sin \left(\frac{2n\pi x}{b-a} \right) dx$$

$$a_o = \frac{2}{2} \int_{-1}^{1} f(x) dx$$

$$\int_{-1}^{-0.5} f(x) dx + \int_{-0.5}^{0.5} f(x) dx + \int_{0.5}^{1} f(x) dx$$

$$0+\int_{-0.5}^{0.5} dx+0$$

$$[x]_{-0.5}^{0.5}$$

$$a_o = 1$$

$$a_n = \frac{2}{2}\int_{-1}^{1} f(x)cos\left(\frac{2n\pi x}{2}\right)dx$$

$$= \int_{-1}^{-0.5}(0)cos(n\pi x)dx + \int_{-0.5}^{-0.5} cos(n\pi x)dx + \int_{0.5}^{1}(0)cos(n\pi x)dx$$

$$= 0+\left[\frac{sin(n\pi x)}{n\pi}\right]_{-0.5}^{0.5}+0$$

$$a_n = \frac{2}{n\pi}sin(\frac{n\pi}{2})$$

$$b_n = \frac{2}{2}\int_{-1}^{1} f(x)sin(\frac{2\pi nx}{2})dx$$

$$= \int_{-1}^{-0.5}(0)sin(n\pi x)dx + \int_{-0.5}^{-0.5} sin(n\pi x)dx + \int_{0.5}^{1}(0)sin(n\pi x)dx$$

As we Know that sin is an odd function, so integration above using property of integration

becomes 0.

$$= 0+0+0$$

$$b_n = 0$$

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

(iii)

$$f(x) = \begin{cases} 0.5 & \text{if} \quad -1 < x < 0 \\ 0.5 & \text{if} \quad 0 < x < 1 \end{cases}$$

$$f(x+2) = f(x)$$

Solution:

$$b-a=2$$

We know that,

$$a_o = \frac{2}{b-a}\int_a^b f(x)\,dx$$
$$a_n = \frac{2}{b-a}\int_a^b f(x)cos\left(\frac{2n\pi x}{b-a}\right)dx$$
$$b_n = \frac{2}{b-a}\int_a^b f(x)sin\left(\frac{2n\pi x}{b-a}\right)dx$$

$$a_0 = \frac{2}{2}\int_{-1}^{1} f(x)dx$$
$$= \int_{-1}^{0}(-0.5)dx + \int_0^1 (0.5)dx$$

$$\frac{-1}{2}[x]_{-1}^{0} + \frac{1}{2}[x]_0^1$$

$$a_0 = 0$$

$$a_n = \frac{2}{2}\int_{-1}^{1} f(x)cos(n\pi x)$$
$$= \int_{-1}^{0}(-0.5)cos(n\pi x)dx + \int_0^1 (0.5)cos(n\pi x)dx$$
$$= \frac{-1}{2}\left[\frac{sin(n\pi x)}{n\pi}\right]_{-1}^{0} + \left[\frac{1}{2}\frac{sin(n\pi x)}{n\pi}\right]_0^1$$
$$= 0+0$$
$$a_n = 0$$

$$b_n = \frac{2}{2}\int_{-1}^{1} f(x)sin(\frac{2n\pi x}{2})$$

$$= \int_{-1}^{0} (-0.5) sin(n\pi x) dx + \int_{0}^{1} (0.5) sin(n\pi x) dx$$

$$= \frac{-1}{2n\pi} [-cos(n\pi x)]_{-1}^{0} + \frac{1}{2n\pi} [-cos(n\pi x)]_{0}^{1}$$

$$= \frac{1}{2n\pi} (1 - cos(n\pi)) + \frac{1}{2n\pi} (1 - cos(n\pi))$$

$$2 * \frac{1}{2n\pi} (1 - cos(n\pi)$$

$$= \frac{1}{n\pi} (1 - cos(n\pi)$$

$$f(x) = \sum_{n=1}^{\infty} \frac{(1 - cos(n\pi)}{n\pi} sin(n\pi x)$$

1.3 Gibbs Phenomenon

The Gibbs phenomenon, is the peculiar manner in which the Fourier series of a piecewise continuously differentiable periodic function behaves at a jump discontinuity. The Gibbs phenomenon involves both the fact that Fourier sums overshoot at a jump discontinuity, and that this overshoot does not die out as more terms are added to the sum. For the one-dimensional case, the simplest mathematical illustration of the Gibbs phenomenon is an approximation of a square wave function.

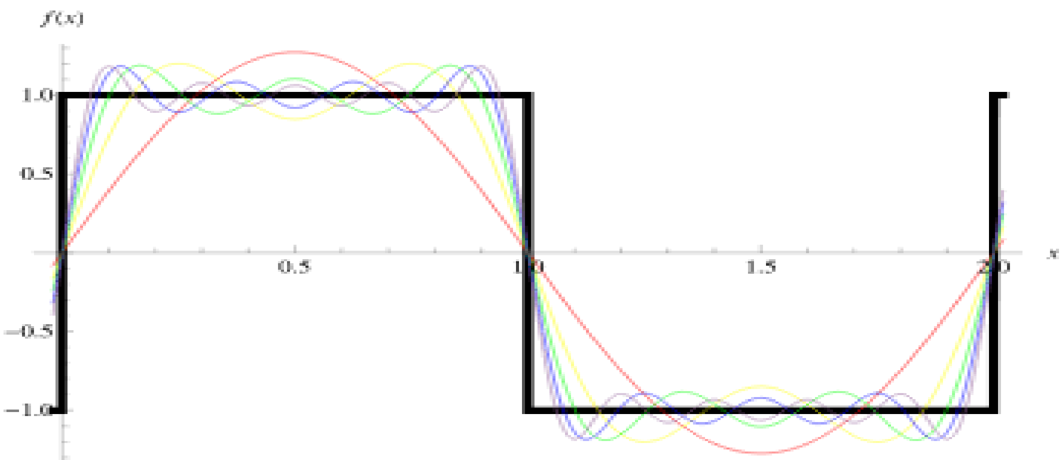

Figure 1: Gibbs Phenomenon

1.4 Half Range Expansion

If a function is defined over half the range, say 0 to L, instead of the full range from -L to L it may be expanded in a series of sine terms only or of cosine terms only. The series produced is then called a half range fourier series.

1.4.1 Even Function and Half Range Cosine Series

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n cos(\frac{n\pi x}{L}) \tag{1}$$

for n=1,2,3,4 ..... where

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

$$a_n = \frac{2}{L}\int_0^L f(x)cos(\frac{n\pi x}{L})dx$$

$$b_n = 0$$

1.4.2 Odd Function and Half Range Sine Series

Since. $a_0 = 0$ and $a_n = 0$,

$$f(x) = \sum_{n=1}^{\infty} b_n sin(\frac{n\pi x}{L}) \tag{2}$$

$$b_n = \frac{2}{L}\int_0^L f(x)sin(\frac{n\pi x}{L})dx$$

1.4.3 $f(x) = x, 0 < x < \pi$

Let f be a function given on the interval (0,a), we define the even/odd extension of f to be even/odd functions on the interval (-a,a), which coincides with f on the half interval (0,a).

Half range even extension

Sketching $f(x) = x$ from $x = 0$ to $x = \pi$:

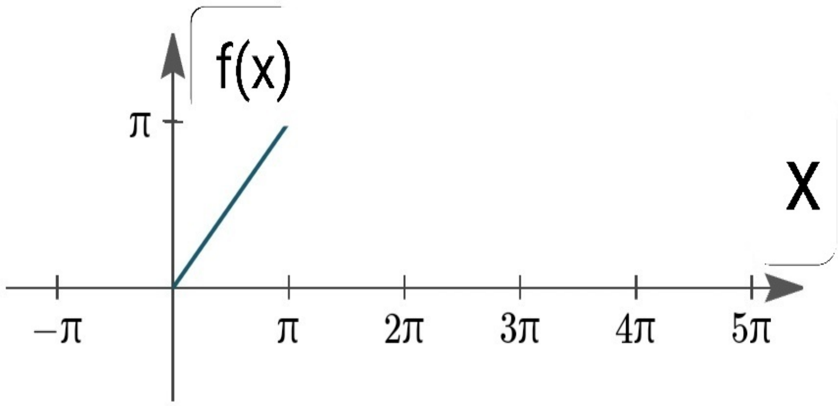

Figure 2: Graph of f(x)

An even function means that it must be symmetrical about the f(x) axis and this is shown in the following figure by the broken line between $t = -\pi$ and $t = 0$.

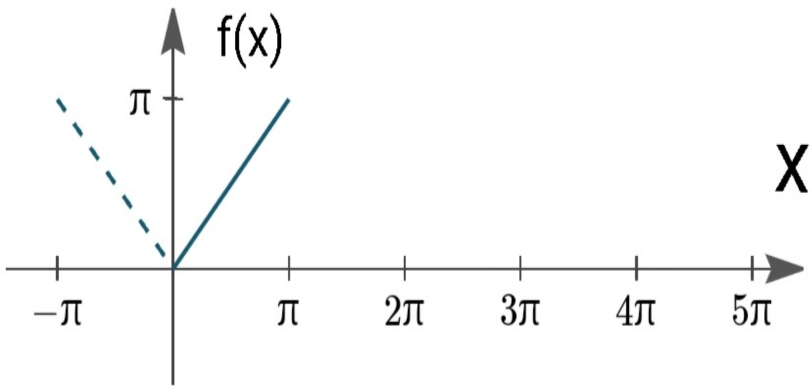

Figure 3: Graph of f(x) showing it as an even function

The "triangular wave form" produced is periodic with period 2π outside of this range as shown by the dotted lines.

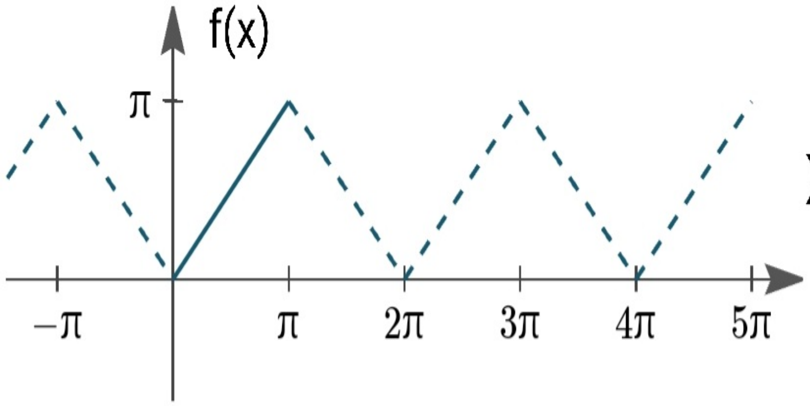

Figure 4: A periodic even function

Half range even extension:

$$f(x) = \begin{cases} -x & \text{if} \quad -\pi \leq x < 0 \\ x & \text{if} \quad 0 \leq x < \pi \end{cases}$$

f(x) is periodic with period 2π

Since the function is even:

$$b_n = 0$$

Here, $L = \pi$

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

$$= \frac{2}{\pi}\int_0^\pi xdx$$

$$= \frac{2}{\pi}\left[\frac{x^2}{2}\right]_2^\pi$$

$$= \frac{2}{\pi}\left[\frac{\pi^2}{2}\right]$$

$$= \pi$$

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

$$= \frac{2}{\pi}\int_0^\pi xcos(nx)dx$$

We know,

$$\int xcos(nx)dx = \frac{1}{n^2}[cos(nx) + nxsin(nx)]$$

$$= \frac{2}{\pi}[\frac{1}{n^2}[cos(nx) + nxsin(nx)]_0^\pi$$

$$= \frac{2}{\pi n^2}[(cos(n\pi) + 0) - (cos0 + 0)]$$

$$= \frac{2}{\pi n^2}[(cos(n\pi) - 1]$$

$$= \frac{2}{\pi n^2}[(-1)^n - 1]$$

$$= \frac{-4}{n^2}$$

(when n is odd)

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n cos(\frac{n\pi x}{L})$$

$$= \frac{\pi}{2} - \frac{4}{\pi} \sum_{n=1}^{\infty} \frac{cos(2n-1)x}{(2n-1)^2}$$

$$\boxed{f(x) = \frac{\pi}{2} - \frac{4}{\pi}(cosx + \frac{1}{9}cos3x + \frac{1}{25}cos5x + ...}$$

Half range odd extension

Sketching $f(x) = x$ from $x = 0$ to $x = \pi$:

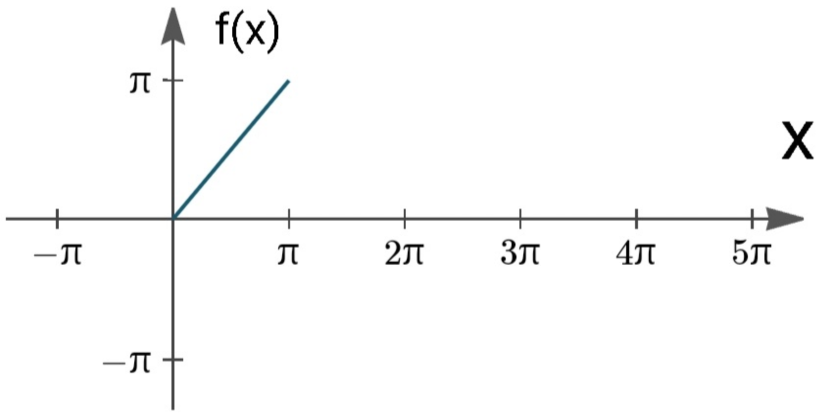

Figure 5: Graph of f(x)

An odd function means that it is symmetrical about the origin and this is shown by the broken lines between $x = \pi$ and $t = 0$

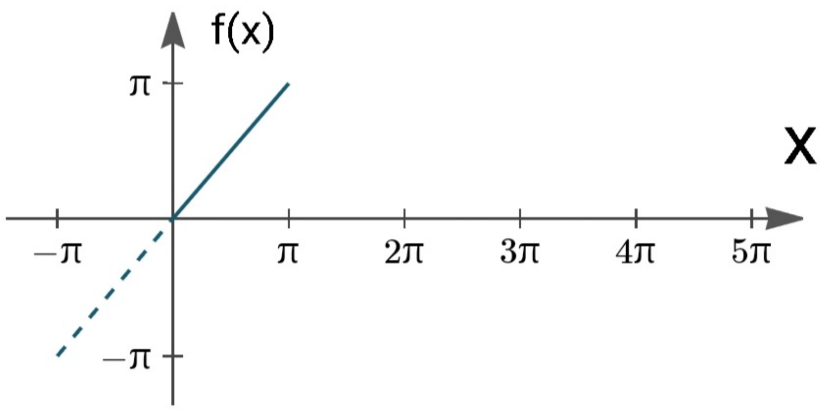

Figure 6: Graph of f(x) showing it as an odd function

The waveform produced is periodic of period 2 outside of this range as shown by the dotted lines.

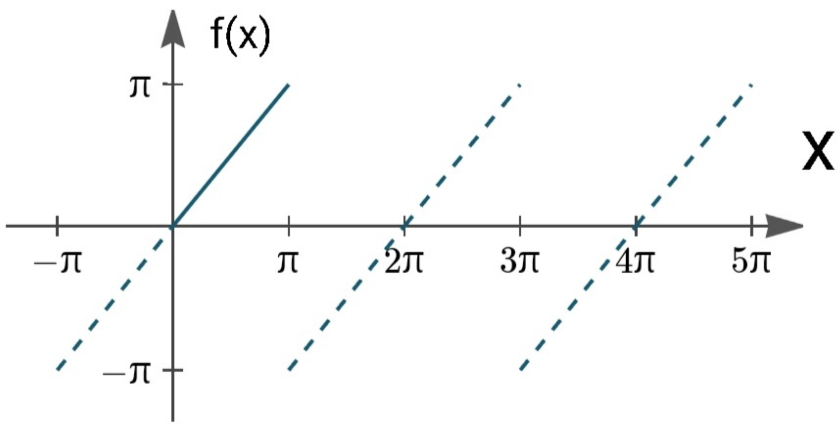

Figure 7: A periodic odd function

Half range even extension:

$$f(x) = \begin{cases} x & \text{if} \quad -\pi \leq x < 0 \\ x & \text{if} \quad 0 \leq x < \pi \end{cases}$$

f(x) is periodic with period 2π

Since the function is odd:

$$a_0 = 0$$

$$a_n = 0$$

Here, $L = \pi$

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

$$= \frac{2}{L} \int_0^\pi x sin(n\pi) dx$$

We know,

$$\int xsin(nx)dx = \frac{1}{n^2}[sin(nx) - nxcos(nx)]$$

$$= \frac{2}{\pi}[\frac{1}{n^2}[sin(nx) - nxcos(nx)]_0^\pi$$

$$= \frac{2}{\pi n^2}[(sin(n\pi) - n\pi cos(n\pi) - (sin0 - 0)]$$

$$= \frac{2}{\pi n^2}[-(-1)^n]$$

$$= -\frac{2}{\pi n^2}(-1)^n$$

$$f(x) = \sum_{n=1}^{\infty} b_n sin(\frac{n\pi x}{L})$$

$$= -\frac{2}{\pi}\sum_{n=1}^{\infty}(-1)^n \frac{sin(nx)}{(n)^2}$$

$$\boxed{f(x) = -\frac{2}{\pi}\left(-sinx + \frac{1}{4}sin2x - .....\right)}$$

Algorithm 1 Fourier Series

function FOURIERCOEFF($f, Prop_Key, a, b, L, N, m_k, tole$)

▷ *f:function*

▷ *$Prop_k$ey :takes the value 0,1,-1 if the function is 'even','odd' or 'neither even nor odd' respectively.*

▷ *a:lower limit ,b:upper limit , tole: tolerance,L:half period,N: numbers of terms to be evaluated*

▷ *m_k: method used for integration takes value 1,2,3 for trapezoidal, simpson and guass-legendre quadrature methods respectively*

if PropKey==1:

for i **do** in range(1,N+1):

fun = lambda x: f(x)*np.sin(i*np.pi*x/L)

if $m_k == 1$:

bn=(1/L)*MyTrap(fun,a,b,1,key1=True,$N_max = 1000, key2 = True, tol = tole)[0]$

elif $m_k == 2$:

bn=(1/L)*MyTrap(fun,a,b,1,key1=True,$N_max = 1000, key2 = True, tol = tole)[0]$

elif $m_k == 3$:

bn=(1/L)*MyLegQuadrature(fun,a,b,10,1,key=True,tol=tole,$m_max = 1000)[0]$

else:

return "Wrong Method Key entered"

$bn_a.append(bn)$

$an_a.append(0)$**return**$a0, an_a, bn_a$

▷ *For even function (that means Propkey=1) we calculate the sum of sine series.*

▷ *The b_n is calculated by the suitable integration method entered by the user and append the values in the list*

▷ *MyTrap is the integration module which evaluate value of the function using trapezoidal method .*

▷ *MySimp is the integration module which evaluate value using simpson method .*

▷ *MyLegQuadrature is the integration module which evaluate using gauss-legendre quadrature.*

def f1(x): ▷ *For odd function (that means Propkey=0) we calculate the sum of cosine series.*

if x>=-np.pi and x<=0: **return** x

elif x>0 and x<=np.pi: **return** x

def pf1(x):

if x>=-np.pi and x<=np.pi : return f1(x) ▷ *The a0 and an is calculated by the suitable integration method entered by the user and append the values in the list.*

elif x>np.pi:

$x_new = x - (np.pi - (-np.pi))$

return pf1(x_new)

elif x<(-np.pi):

$x_new = x + (np.pi - (-np.pi))$

return pf1(x_new)

▷ *For neither odd or even function we calculate the sum of cosine and sine series.*

▷ *The a0,bn and an is calculated by the suitable integration method and append values in list.*

2(b)

I think Gauss-legendre Quadrature is the most appropriate numerical method for the above functions in previous questions.This method is optimal for all polynomials of degree less or equal to 2n-1, while the newton-cotes(trapezoidal and simpson) formulas are optimal for all polynomials of degree less than or equal to n.

But for above functions gauss-legendre quadrature is better as the integral is very close to true value and the absolute error is very close to zero. And this intuition has verified when we'd checked the value of integral by other numerical method.

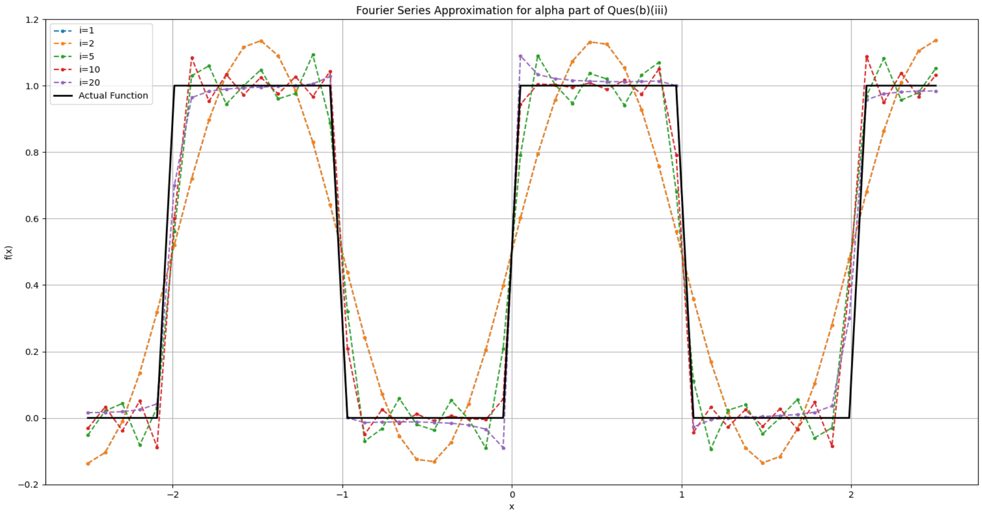

Figure 8: *Fourier Series Approximation for* α *using Gauss Legendre Quadrature*

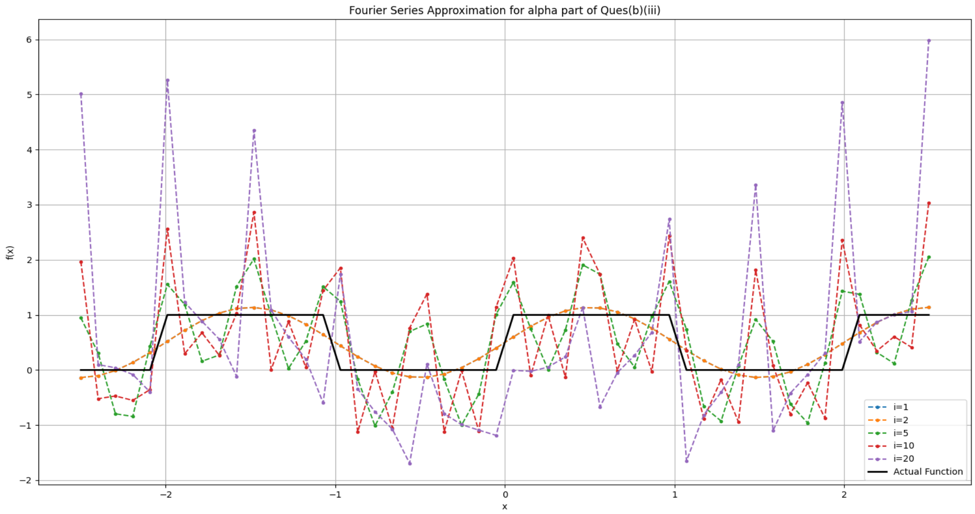

Figure 9: *Fourier Series Approximation for using Simpson Method*

If we had to approximate the output of a full wave rectifier by Fourier series, Gauss Legendre Quadrature would be most appropriate because for finding the output we have to integrate sine terms and Quadrature will serve the purpose better than simpson and trapezoidal method.

2 RESULTS AND DISCUSSION

For

$$f(x) = \begin{cases} 0 & \text{if} \quad -1 < x < 0 \\ 1 & \text{if} \quad 0 < x < 1 \end{cases}$$

$$f(x+2) = f(x)$$

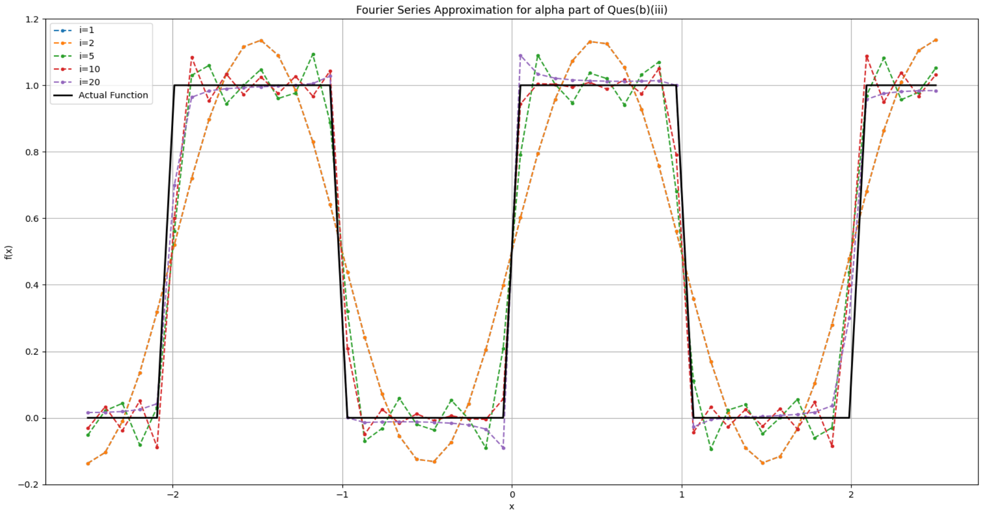

Figure 10: *Fourier Series Approximation for* α

For

$$f(x) = \begin{cases} 0 & \text{if} \quad -1 < x < -0.5 \\ 1 & \text{if} \quad -0.5 < x < 0.5 \\ 0 & \text{if} \quad 0.5 < x < 1 \end{cases}$$

$$f(x+2) = f(x)$$

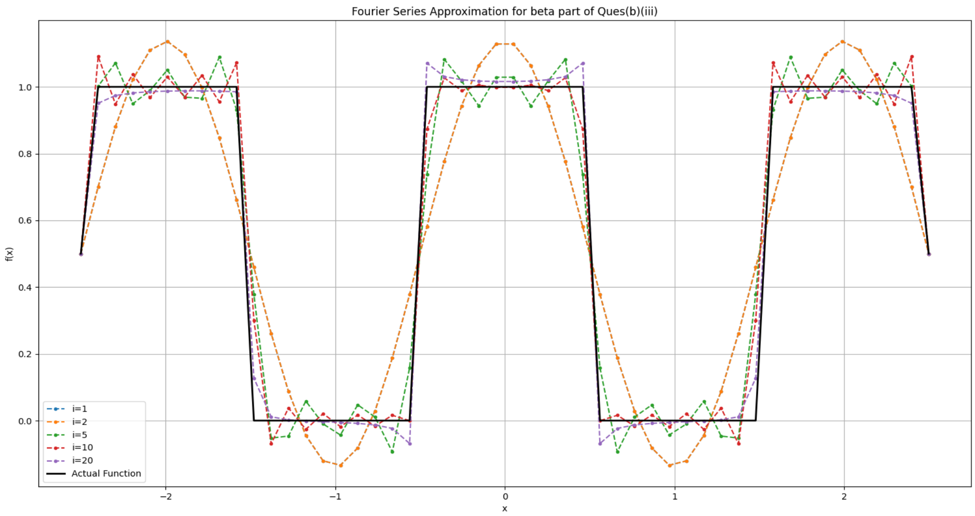

Figure 11: *Fourier Series Approximation for* β

For

$$f(x) = \begin{cases} -0.5 & \text{if} \quad -1 < x < 0 \\ 0.5 & \text{if} \quad 0 < x < 1 \end{cases}$$

$$f(x+2) = f(x)$$

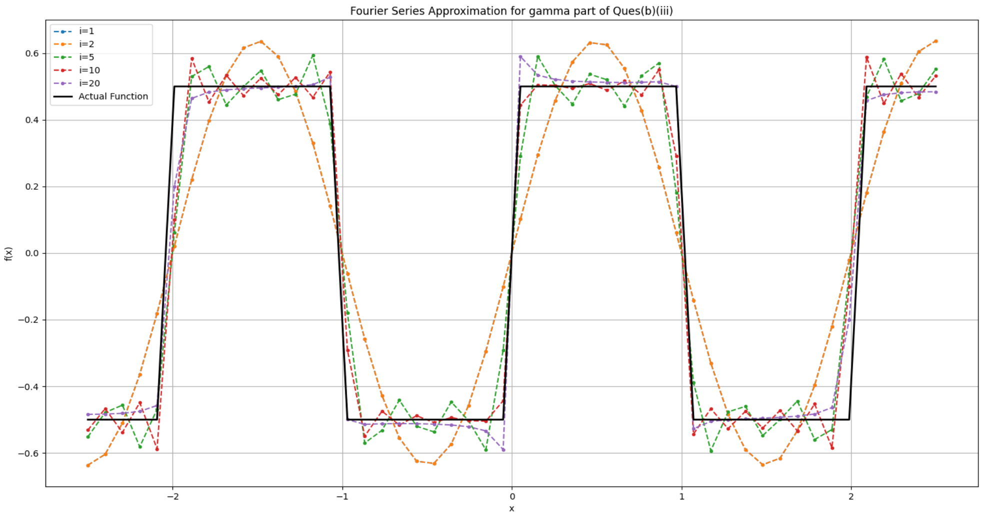

Figure 12: *Fourier Series Approximation for* γ

From the above graphs we can see the partial sums after one term,then two terms,then five terms,then ten terms,and then twenty terms. We can see the Gibbs phenomenon appearing as these "partial sums" include more terms.Away from the jumps,we safely approach $f(x)$=1 or -1 for first two graphs and $f(x)$=0.5 or -0.5 for third graph.
The Gibbs phenomenon is the overshoot that moves closer and closer to the jumps. We can clearly see that the overshoot height goes above 1 or -1 and it does not decrease with more terms of the series!.Thus we can say that overshoot is present in all discontinuous functions.

For

$$f(x) = x \quad \text{for} \quad 0 < x < \pi$$

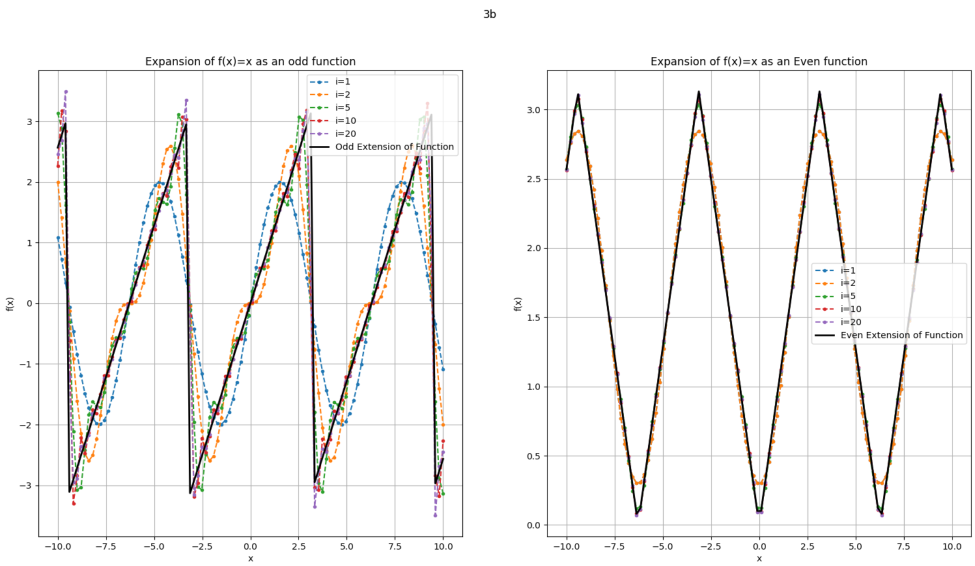

Figure 13: *Half Range Expansion of Fourier series*

Above plot shows the f(x) vs x plot for half range sine and cosine expansions.Series calculated for N=20 is very similar to the extended function both as odd as well as even function.

A Programs

```
#Name= Monu Chaurasiya
#College Roll No. = 2020 PHY1102
#University Roll No. = 20068567035

'''
PARTNERS:
#Name= Pawanpreet Kaur
#College Roll No. = 2020 PHY1092
#University Roll No. = 20068567038

#Name= Prateek Bhardwaj
#College Roll No. = 2020 PHY1110
#University Roll No. = 20068567042
'''

from MyIntegration import MySimp
from MyIntegration import MyTrap
from MyIntegration import MyLegQuadrature
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import integrate
from sympy import *
from sympy import simplify

x=symbols('x')

#Function for Fourier Coefficient
def FourierCoeff(f,Prop_Key,a,b,L,N,m_k,tole):
    an=0;an_a=[]
    bn=0;bn_a=[]
    a0=0
    if Prop_Key==1:
        for i in range(1,N+1):
            fun = lambda x: f(x)*np.sin(i*np.pi*x/L)
            if m_k==1:
                bn=(1/L)*MyTrap(fun,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
            elif m_k==2:
                bn=(1/L)*MyTrap(fun,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
            elif m_k==3:
                 bn=(1/L)*MyLegQuadrature(fun,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
            else:
                 return "Wrong Method Key entered"
            bn_a.append(bn)
            an_a.append(0)
        return a0,an_a,bn_a

    elif Prop_Key==0:
```

```
     if m_k==1:
         a0=(1/L)*MyTrap(f,a,b,1,key1=True,N_max=1000,key2=True,tol=0.1e-3)[0]
     elif m_k==2:
         a0=(1/L)*MySimp(f,a,b,2,key1=True,N_max=1000,key2=True,tol=tole)[0]
     elif m_k==3:
         a0=(1/L)*MyLegQuadrature(f,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
     else:
         return "Wrong Method Key entered"

     for i in range(1,N+1):
         fun = lambda x: f(x)*np.cos(i*np.pi*x/L)
         if m_k==1:
            an=(1/L)*MyTrap(fun,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
         elif m_k==2:
            an=(1/L)*MySimp(fun,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
         elif m_k==3:
             an=(1/L)*MyLegQuadrature(fun,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
         else:
             return "Wrong Method Key entered"
         an_a.append(an)
         bn_a.append(0)
     return a0,an_a,bn_a

 elif Prop_Key==-1:
     if m_k==1:
         a0=(1/L)*MyTrap(f,a,b,1,key1=True,N_max=1000,key2=True,tol=0.1e-3)[0]
     elif m_k==2:
         a0=(1/L)*MySimp(f,a,b,2,key1=True,N_max=1000,key2=True,tol=tole)[0]
     elif m_k==3:
         a0=(1/L)*MyLegQuadrature(f,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
     else:
         return "Wrong Method Key entered"

     for i in range(1,N+1):
         fun1 = lambda x: f(x)*np.cos(i*np.pi*x/L)
         fun2 = lambda x: f(x)*np.sin(i*np.pi*x/L)
         if m_k==1:
            an=(1/L)*MyTrap(fun1,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
            bn=(1/L)*MyTrap(fun2,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
         elif m_k==2:
            an=(1/L)*MySimp(fun1,a,b,2,key1=True,N_max=1000,key2=True,tol=tole)[0]
            bn=(1/L)*MySimp(fun2,a,b,2,key1=True,N_max=1000,key2=True,tol=tole)[0]
```

```
            elif m_k==3:
                an=(1/L)*MyLegQuadrature(fun1,a,b,10,1,key=True,tol=
    tole,m_max=1000)[0]
                bn=(1/L)*MyLegQuadrature(fun2,a,b,10,1,key=True,tol=
    tole,m_max=1000)[0]
            else:
                return "Wrong Method Key entered"
            an_a.append(an)
            bn_a.append(bn)
        return a0,an_a,bn_a
    else :
        return "Wrong key entered"


def Task(L,f1,pf1,Prop_Key,m_k,tit,filename1,filename2,filename3,
    filename4,filename5):
    N_a=[1,2,5,10, 20]
    x_a=np.linspace(-2.5,2.5,50)
    ex=[]
    f_v=[]
    f_=[]
    a_c=[]
    b_c=[]
    E1=[];E2=[];E3=[];E4=[];E5=[]
    A_X=[-0.5,0,0.5]
    ex2=[]
    for x in x_a:
        ex.append(pf1(x))
    for x in A_X:
        ex2.append(pf1(x))
    for N in N_a:
         s=FourierCoeff(pf1,Prop_Key=Prop_Key,a=-1*L,b=1*L,L=1,N=N,m_k
    =m_k,tole=0.1e-8)
         a0=s[0]
         #print(a0)
         e=s[1]
         e1=s[2]
         d_a=[]
         d_=[]
         for x in A_X:
             sin=0
             cos=0

             for (i,an,bn) in zip(range(1,N+1),e,e1):
                 sin+=bn*np.sin(i*np.pi*x/L)
                 cos+=an*np.cos(i*np.pi*x/L)
             d=a0/2+sin+cos
             d_.append(d)
         f_.append(d_)
         d_=[]

         for x in x_a:
             sin=0
             cos=0

```

```
            for (i,an,bn) in zip(range(1,N+1),e,e1):
                sin+=bn*np.sin(i*np.pi*x/L)
                cos+=an*np.cos(i*np.pi*x/L)
            d=a0/2+sin+cos
            d_a.append(d)
        f_v.append(d_a)
        d_a=[]
        a_c.append(e)
        b_c.append(e1)

DataOut1 = np.column_stack((a_c[0],b_c[0]))
np.savetxt(filename1, DataOut1,delimiter=',')
DataOut2 = np.column_stack((a_c[1],b_c[1]))
np.savetxt(filename2, DataOut2,delimiter=',')
DataOut3 = np.column_stack((a_c[2],b_c[2]))
np.savetxt(filename3, DataOut3,delimiter=',')
DataOut4 = np.column_stack((a_c[3],b_c[3]))
np.savetxt(filename4, DataOut4,delimiter=',')
DataOut5 = np.column_stack((a_c[4],b_c[4]))
np.savetxt(filename5, DataOut5,delimiter=',')

plt.plot(x_a,f_v[0],marker=".",label="i=1",linestyle='dashed')
plt.plot(x_a,f_v[1],marker=".",label="i=2",linestyle='dashed')
plt.plot(x_a,f_v[2],marker=".",label="i=5",linestyle='dashed')
plt.plot(x_a,f_v[3],marker=".",label="i=10",linestyle='dashed')
plt.plot(x_a,f_v[4],marker=".",label="i=20",linestyle='dashed')
plt.plot(x_a,ex,linewidth=2,c="black",label="Actual Function")
plt.grid()
plt.title(tit)
plt.legend()
plt.xlabel("x")
plt.ylabel("f(x)")
plt.show()
print()
print("---------------Value of f(x) calculated using Fourier Expansion------------------")
data={"x_a":x_a,"f(x)(i=1)":f_v[0],"f(x)(i=2)":f_v[1],"f(x)(i=5)":f_v[2],"f(x)(i=10)":f_v[3],"f(x)(i=20)":f_v[4],"f(x)(exact)":ex}
print(pd.DataFrame(data))



for (r,t,y,p,j,l) in zip (f_[0],f_[1],f_[2],f_[3],f_[4],range(3)):
    E1.append(abs(r-ex2[l]))
    E2.append(abs(t-ex2[l]))
    E3.append(abs(y-ex2[l]))
    E4.append(abs(p-ex2[l]))
    E5.append(abs(j-ex2[l]))

print()
print("----------------Absolute error for x=[-0.5,0,0.5]------------------")
data={"x":A_X,"Error(i=1)":E1,"Error(i=2)":E2,"Error(i=5)":E3,"Error(i=10)":E4,"Error(i=20)":E5}
print(pd.DataFrame(data))
```

```
#Function1
def f1(x):
    if x>-1 and x<0:
        return 0
    elif x>0 and x<1:
        return 1
    elif x==-1 or x==0 or x==1:
        return 1/2

def pf1(x):
    if x>=-1 and x<=1 :
        return f1(x)
    elif x>1:
        x_new=x-(1-(-1))
        return pf1(x_new)
    elif x<(-1):
        x_new=x+(1-(-1))
        return pf1(x_new)

#Function2
def f2(x):
    if x>-1 and x<-0.5:
        return 0
    elif x>-0.5 and x<0.5:
        return 1
    elif x>0.5 and x<1:
        return 0
    elif x==-1 or x==0.5 or x==1 or x==-0.5:
        return 1/2

def pf2(x):
    if x>=-1 and x<=1 :
        return f2(x)
    elif x>1:
        x_new=x-(1-(-1))
        return pf2(x_new)
    elif x<(-1):
        x_new=x+(1-(-1))
        return pf2(x_new)

#Function3
def f3(x):
    if x>-1 and x<0:
        return -0.5
    elif x>0 and x<1:
        return 0.5
    elif x==-1 or x==0 or x==1:
        return 0

def pf3(x):
    if x>=-1 and x<=1 :
        return f3(x)
    elif x>1:
        x_new=x-(1-(-1))
```

```
            return pf3(x_new)
        elif x<(-1):
            x_new=x+(1-(-1))
            return pf3(x_new)

print("********************************(ALPHA)
    ********************************************")
Task(1,f1,pf1,-1,3,"Fourier Series Approximation for alpha part of
    Ques(b)(iii)","alpha1.dat","alpha2.dat","alpha3.dat","alpha4.dat","
    alpha5.dat")
print()
print("********************************(BETA)
    ********************************************")
Task(1,f2,pf2,0,3,"Fourier Series Approximation for beta part of Ques(
    b)(iii)","beta1.dat","beta2.dat","beta3.dat","beta4.dat","beta5.dat
    ")
print()
print("********************************(GAMMA)
    ********************************************")
Task(1,f3,pf3,1,3,"Fourier Series Approximation for gamma part of Ques
    (b)(iii)","gamma1.dat","gamma2.dat","gamma3.dat","gamma4.dat","
    gamma5.dat")
```

Source Code 1: Python Program

```
#Name= Monu Chaurasiya
#College Roll No. = 2020 PHY1102
#University Roll No. = 20068567035
'''
PARTNERS:
#Name= Pawanpreet Kaur
#College Roll No. = 2020 PHY1092
#University Roll No. = 20068567038

#Name= Prateek Bhardwaj
#College Roll No. = 2020 PHY1110
#University Roll No. = 20068567042
'''

from MyIntegration import MySimp
from MyIntegration import MyTrap
from MyIntegration import MyLegQuadrature
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import integrate
from sympy import *
from sympy import simplify

x=symbols('x')

#Function for Fourier Coefficient
def FourierCoeff(f,Prop_Key,a,b,L,N,m_k,tole):
    an=0;an_a=[]
    bn=0;bn_a=[]
```

```
    a0=0
    if Prop_Key==1:
        for i in range(1,N+1):
            fun = lambda x: f(x)*np.sin(i*np.pi*x/L)
            if m_k==1:
                bn=(1/L)*MyTrap(fun,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
            elif m_k==2:
                bn=(1/L)*MyTrap(fun,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
            elif m_k==3:
                 bn=(1/L)*MyLegQuadrature(fun,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
            else:
                return "Wrong Method Key entered"
            bn_a.append(bn)
            an_a.append(0)
        return a0,an_a,bn_a

    elif Prop_Key==0:
        if m_k==1:
            a0=(1/L)*MyTrap(f,a,b,1,key1=True,N_max=1000,key2=True,tol=0.1e-3)[0]
        elif m_k==2:
            a0=(1/L)*MySimp(f,a,b,2,key1=True,N_max=1000,key2=True,tol=tole)[0]
        elif m_k==3:
            a0=(1/L)*MyLegQuadrature(f,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
        else:
            return "Wrong Method Key entered"

        for i in range(1,N+1):
            fun = lambda x: f(x)*np.cos(i*np.pi*x/L)
            if m_k==1:
                an=(1/L)*MyTrap(fun,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
            elif m_k==2:
                an=(1/L)*MySimp(fun,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
            elif m_k==3:
                 an=(1/L)*MyLegQuadrature(fun,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
            else:
                return "Wrong Method Key entered"
            an_a.append(an)
            bn_a.append(0)
        return a0,an_a,bn_a

    elif Prop_Key==-1:
        if m_k==1:
            a0=(1/L)*MyTrap(f,a,b,1,key1=True,N_max=1000,key2=True,tol=0.1e-3)[0]
        elif m_k==2:
            a0=(1/L)*MySimp(f,a,b,2,key1=True,N_max=1000,key2=True,tol
```

```
=tole)[0]
        elif m_k==3:
            a0=(1/L)*MyLegQuadrature(f,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
        else:
            return "Wrong Method Key entered"

        for i in range(1,N+1):
            fun1 = lambda x: f(x)*np.cos(i*np.pi*x/L)
            fun2 = lambda x: f(x)*np.sin(i*np.pi*x/L)
            if m_k==1:
               an=(1/L)*MyTrap(fun1,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
               bn=(1/L)*MyTrap(fun2,a,b,1,key1=True,N_max=1000,key2=True,tol=tole)[0]
            elif m_k==2:
               an=(1/L)*MySimp(fun1,a,b,2,key1=True,N_max=1000,key2=True,tol=tole)[0]
               bn=(1/L)*MySimp(fun2,a,b,2,key1=True,N_max=1000,key2=True,tol=tole)[0]
            elif m_k==3:
                an=(1/L)*MyLegQuadrature(fun1,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
                bn=(1/L)*MyLegQuadrature(fun2,a,b,10,1,key=True,tol=tole,m_max=1000)[0]
            else:
                return "Wrong Method Key entered"
            an_a.append(an)
            bn_a.append(bn)
        return a0,an_a,bn_a
    else :
        return "Wrong key entered"


def Task(L,f1,pf1,Prop_Key,m_k):
    N_a=[1,2,5,10, 20]
    x_a=np.linspace(-10,10,100)
    f_v=[]
    f_=[]
    a_c=[]
    b_c=[]
    E1=[];E2=[];E3=[];E4=[];E5=[]
    A_X=[0,np.pi/2,np.pi]
    ex2=[]
    for x in A_X:
        ex2.append(pf1(x))
    for N in N_a:
         s=FourierCoeff(pf1,Prop_Key=Prop_Key,a=0,b=L,L=np.pi,N=N,m_k=m_k,tole=0.1e-8)
         a0=2*s[0]
         e=np.array(s[1])
         e=np.dot(2,e)
         e1=np.array(s[2])
         e1=np.dot(2,e1)
         d_a=[]
```

```python
         d_=[]
         ex=[]
         for x in x_a:
             ex.append(pf1(x))
         for x in A_X:
             sin=0
             cos=0

             for (i,an,bn) in zip(range(1,N+1),e,e1):
                 sin+=bn*np.sin(i*np.pi*x/L)
                 cos+=an*np.cos(i*np.pi*x/L)
             d=a0/2+sin+cos
             d_.append(d)
         f_.append(d_)
         d_=[]

         for x in x_a:
             sin=0
             cos=0

             for (i,an,bn) in zip(range(1,N+1),e,e1):
                 sin+=bn*np.sin(i*np.pi*x/L)
                 cos+=an*np.cos(i*np.pi*x/L)
             d=a0/2+sin+cos
             d_a.append(d)
         f_v.append(d_a)
         d_a=[]
         a_c.append(e)
         b_c.append(e1)

    print()
    print("---------------Value of f(x) calculated using Fourier Expansion------------------")
    data={"x_a":x_a,"f(x)(i=1)":f_v[0],"f(x)(i=2)":f_v[1],"f(x)(i=5)":f_v[2],"f(x)(i=10)":f_v[3],"f(x)(i=20)":f_v[4],"f(x)(exact)":ex}
    print(pd.DataFrame(data))



    for (r,t,y,p,j,l) in zip (f_[0],f_[1],f_[2],f_[3],f_[4],range(3)):
        E1.append(abs(r-ex2[l]))
        E2.append(abs(t-ex2[l]))
        E3.append(abs(y-ex2[l]))
        E4.append(abs(p-ex2[l]))
        E5.append(abs(j-ex2[l]))

    print()
    print("----------------Absolute error for x=[0,pi/2,pi]------------------")
    data={"x":A_X,"Error(i=1)":E1,"Error(i=2)":E2,"Error(i=5)":E3,"Error(i=10)":E4,"Error(i=20)":E5}
    print(pd.DataFrame(data))
    return x_a,f_v

#odd Function
```

```
def f1(x):
    if x>=-np.pi and x<=0:
        return x
    elif x>0 and x<=np.pi:
        return x


def pf1(x):
    if x>=-np.pi and x<=np.pi :
        return f1(x)
    elif x>np.pi:
        x_new=x-(np.pi-(-np.pi))
        return pf1(x_new)
    elif x<(-np.pi):
        x_new=x+(np.pi-(-np.pi))
        return pf1(x_new)

#even function
def f2(x):
    if x>=-np.pi and x<=0:
        return -1*x
    elif x>0 and x<=np.pi:
        return x


def pf2(x):
    if x>=-np.pi and x<=np.pi :
        return f2(x)
    elif x>np.pi:
        x_new=x-(np.pi-(-np.pi))
        return pf2(x_new)
    elif x<(-np.pi):
        x_new=x+(np.pi-(-np.pi))
        return pf2(x_new)

x_a1,f_v1=Task(np.pi,f1,pf1,1,3)
x_a=np.linspace(-10,10,100)
s1=[]
for x in x_a:
    s1.append(pf1(x))
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('3b')
ax1.plot(x_a1,f_v1[0],marker=".",label="i=1",linestyle='dashed')
ax1.plot(x_a1,f_v1[1],marker=".",label="i=2",linestyle='dashed')
ax1.plot(x_a1,f_v1[2],marker=".",label="i=5",linestyle='dashed')
ax1.plot(x_a1,f_v1[3],marker=".",label="i=10",linestyle='dashed')
ax1.plot(x_a1,f_v1[4],marker=".",label="i=20",linestyle='dashed')
ax1.plot(x_a1,s1,linewidth=2,c="black",label="Odd Extension of Function")
ax1.grid()
ax1.set(xlabel="x",ylabel="f(x)",title="Expansion of f(x)=x as an odd function")
ax1.legend()


```

```
x_a,f_v=Task(np.pi,f2,pf2,0,3)
x_a=np.linspace(-10,10,100)
s=[]
for x in x_a:
    s.append(pf2(x))

ax2.plot(x_a,f_v[0],marker=".",label="i=1",linestyle='dashed')
ax2.plot(x_a,f_v[1],marker=".",label="i=2",linestyle='dashed')
ax2.plot(x_a,f_v[2],marker=".",label="i=5",linestyle='dashed')
ax2.plot(x_a,f_v[3],marker=".",label="i=10",linestyle='dashed')
ax2.plot(x_a,f_v[4],marker=".",label="i=20",linestyle='dashed')
ax2.plot(x_a,s,linewidth=2,c="black",label="Even Extension of Function")
ax2.grid()
ax2.set(xlabel="x",ylabel="f(x)",title="Expansion of f(x)=x as an Even function")
ax2.legend()
plt.show()
```

Source Code 2: Python Program