Fitting of Functions

Arunima Sarkar, EE21S062

1 Question (1)

The identity given:

$$\frac{d}{dx}(xJ_1(x)) = xJ_0(x) \tag{1}$$

The function for this is:

```
def func(x):
    return x*sp.jv(1,x)
```

The function given in equation (1) is fitted using Chebyshev Polynomials, such that

$$f(x) = \sum_{i=0}^{m-1} c_i T_i(x) \tag{2}$$

Here T_i are the Chebyshev polynomials of order i and the entire series is approximated by m terms.

1.1 The function used to generate the coefficients is:

```
def chebft(a,b,n):
    bma=0.5*(b-a)
    bpa=0.5*(b+a)
    f=[]
    c=[]
    for k in range(1,n+1):
        y=np.cos(np.pi*(k-0.5)/n)
        f.append(func(y*bma+bpa))
    fac=2/n
    for j in range(n):
        summation=0
        for k in range(1,n+1):
            summation += f[k-1]*np.cos(np.pi*j*(k-0.5)/n)
        c.append(fac*summation)
    return c
```

The corresponding plot is:

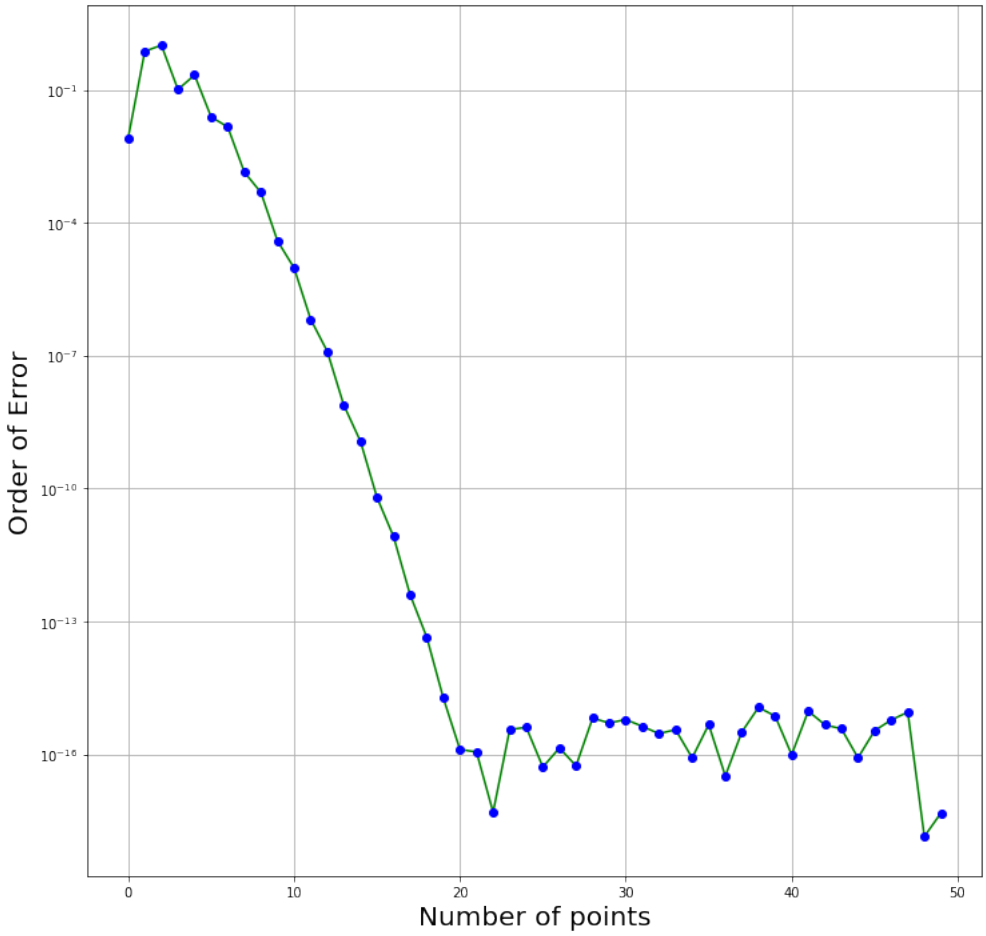

Figure 1: Number of points Vs Error order

As we can see that around m=20, the error is close to machine precision, thus 20 coefficients are taken from now on for approximation of this function.

1.2 Chebyshev Approximation

Thus the Chebyshev Approximation created is using the following function,

```
coeffients=chebft(0,5,50)
def chebyshev_poly(n,x): #Calculating the Tn(x) by forward recursion
    if n==0:
        return 1
```

```
    if n==1:
        return x
    else:
        return 2*x*chebyshev_poly(n-1,x)-chebyshev_poly(n-2,x)

def function_from_chebyshev(m,x,coeff,a,b):
    function=0
    y=-1+(x-a)*2/(b-a)
    for k in range(m):
        function+=coeff[k]*chebyshev_poly(k,y)

    function=function-(0.5*coeff[0])
    return function

def error1():
    error=[]
    x=np.linspace(0,5,20)
    for i in range(20):
        error.append(np.abs(func(x[i])-function_from_chebyshev(20,x[i],
                    coeffients,0,5)))
    return error
```

The error plot is:

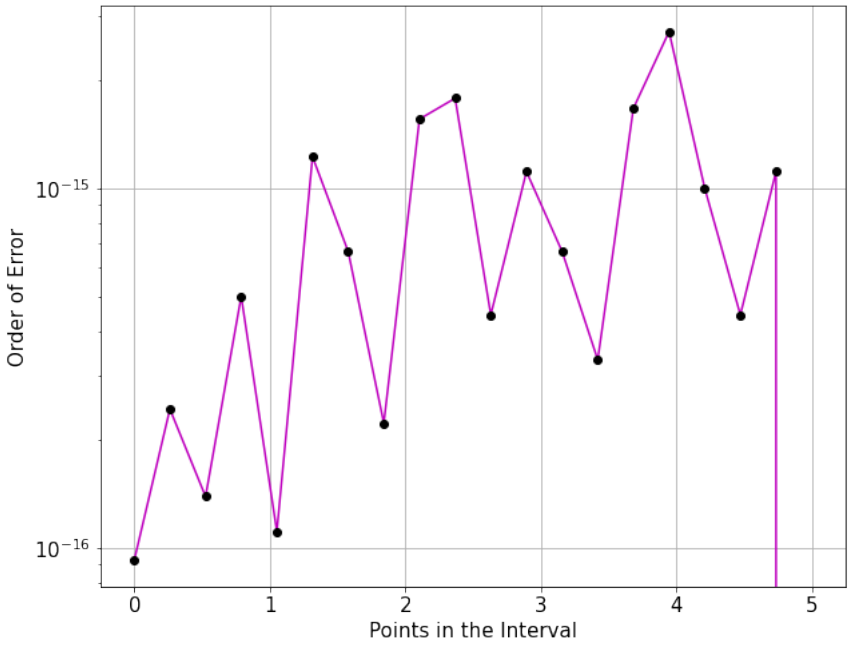

Figure 2: Points in range(0,5) Vs Error order

We can clearly see, that the max error is close to 10^{-14}, thus Chebyshev way of approximating is very useful, with just 20 terms.

1.3 Chebyshev sum for the derivatives

I have calculated the coefficients using the identity below,

$$c_{i-1}^{'} = c_{i+1}^{'} + 2ic_i \qquad (i = m-1, m-2, ...1) \tag{3}$$

Here c_i's are the coefficients of $f(x)$ and $c_i^{'}$ are coefficients of $f^{'}(x)$.

Code

```
def function_derivative(x):
    return x*sp.jv(0,x)

def chder(a,b,coeff,m):
    coeff_der=[0]*m
    coeff_der[-2]=2*(m-1)*coeff[m-1]
    j=m-3
    while j>=0:
        coeff_der[j]=coeff_der[j+2]+2*(j+1)*coeff[j+1]
        j-=1
    normalized_list=[value*2/(b-a) for value in coeff_der]
    return normalized_list

derivative_coeff=chder(0,5,coeffients,20)

def error_derivative():
    error=[]
    x=np.linspace(0,5,20)
    for i in range(20):
        error.append(np.abs(function_derivative(x[i])-
        function_from_chebyshev(20,x[i],derivative_coeff,0,5)))
    return error
```

The corresponding plot is,:

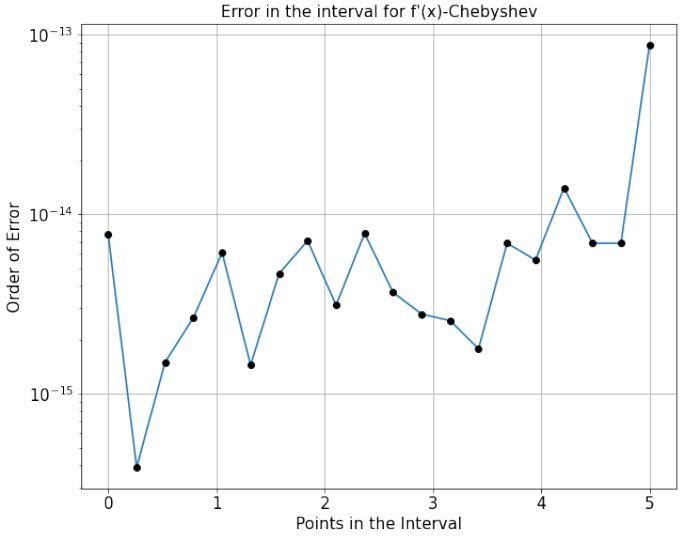

Figure 3: Error plot for Chebyshev fit

Figure 4: Error plot for Difference of samples

We can clearly see that Chebyshev provides a better fit, which is of order 10^{-14} compared to Difference of samples that gives an error around 10^{-10}, with delta=10^{-5}.

Computational Cost of Difference of Samples: The Taylor expansion of a function f(x) for Δx is,

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + (\Delta x)^2 \frac{f''(x)}{2!} + ... \tag{4}$$

From the centered difference approach ,

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + (\Delta x)^2 \frac{f''(x)}{2!} + (\Delta x)^3 \frac{f'''(x)}{3!} + ... \tag{5}$$

$$f(x-\Delta x) = f(x) - \Delta x f'(x) + (\Delta x)^2 \frac{f''(x)}{2!} - (\Delta x)^3 \frac{f'''(x)}{3!} + ... \tag{6}$$

gives $f'(x)$ as,

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} + \Delta x^2 \frac{f'''(x)}{3!} + ... \tag{7}$$

But from the definition of limits, the terms from Δx^2 on-wards contribute to the error,thus larger the Δx larger will be it's contribution to the error. Thus reducing Δx will reduce error , but that too will hit a bottleneck when smaller Δx will make the $f(x+\Delta x) - f(x-\Delta x)$ term inaccurate, thus there will be a certain sweet spot for Δx which in our case is 10^{-5} for an error of order 10, as anything below or above it will give higher order error.
However the cost is low, one just needs two function computation and 1 division, but accuracy is low.

Computational Cost of Chebyshev Fitting: With Chebyshev just sufficient coefficients have to be computed that gives machine level precision.But one needs to calculate the m coefficients of its derivatives as shown in equation 3, and m multiplications and $m-1$ additions to compute the derivative of the function. Thus its computational cost is high, but it gives quiet accurate answers.

2 Question (2)

The functions to be approximated are:

$$f(x) = \exp(x) \tag{8}$$

$$g(x) = \frac{1}{x^2 + \delta^2} \tag{9}$$

$$h(x) = \frac{1}{\cos^2(\pi x/2) + \delta^2} \tag{10}$$

$$u(x) = \exp(-|x|) \tag{11}$$

$$v(x) = \sqrt{x + 1.1} \tag{12}$$

The code is:

```
def get_d():
    return 3

def f(x):
    return math.exp(x)

def g(x):
    d=get_d()
    return 1/(x**2+ d**2)

def h(x):
    d=get_d()
    return 1/((math.cos(math.pi*x/2))**2+d**2)

def u(x):
    return math.exp(-1*abs(x))

def v(x):
    return math.sqrt(x+1.1)


def chebft_modified(a,b,n,func):
    bma=0.5*(b-a)
    bpa=0.5*(b+a)
    f=[]
    c=[]

    for k in range(1,n+1):
        y=np.cos(np.pi*(k-0.5)/n)
        f.append(func(y*bma+bpa))
    fac=2/n
```

```
    for j in range(n):
        summation=0
        for k in range(1,n+1):
            summation += f[k-1]*np.cos(np.pi*j*(k-0.5)/n)
        c.append(fac*summation)
    return c

coeff_f=chebft_modified(-1,1,100,f)
coeff_v=chebft_modified(-1,1,100,v)
coeff_g=chebft_modified(-1,1,100,g)
coeff_h=chebft_modified(-1,1,100,h)
```

The plot for the coefficients of all but u(x) is:

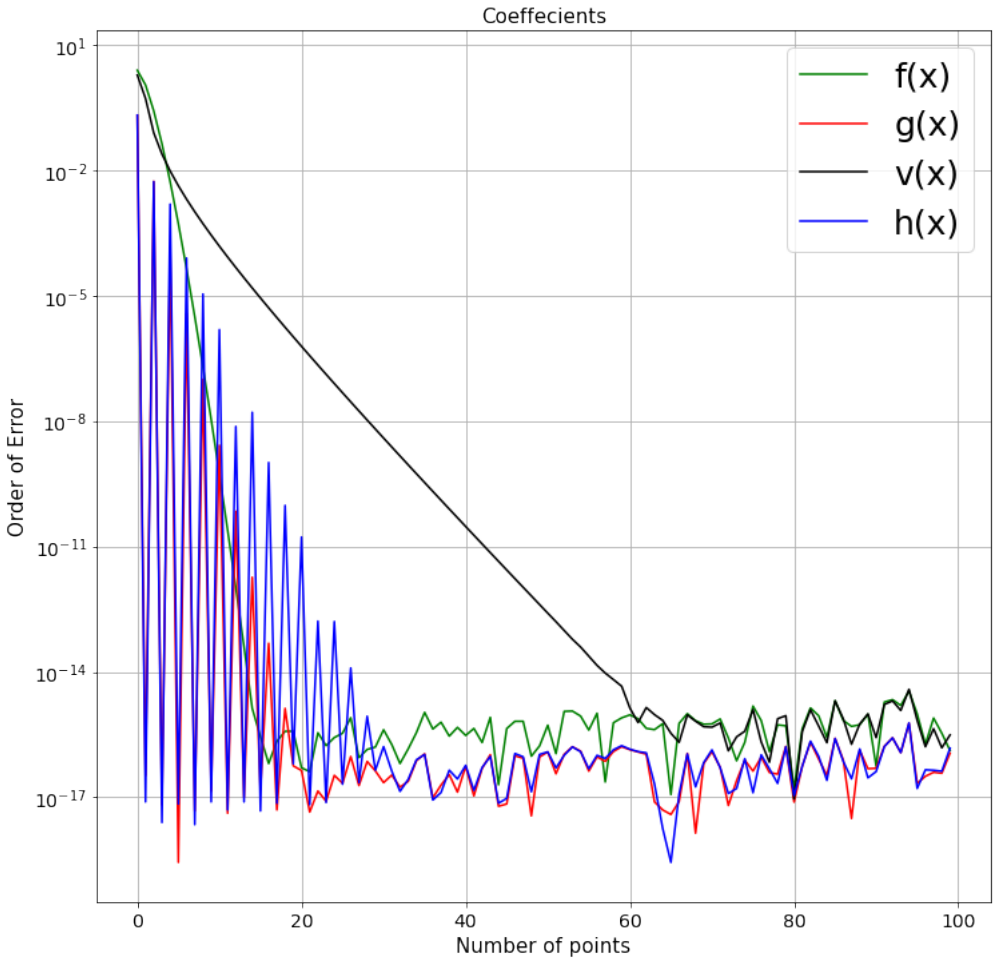

Figure 5: Plot for coefficients for different number of points

Function f(x) requires 18 coefficients, g(x) requires 20 coefficients, h(x) requires 30 coefficients, v(x) requires 60 coefficients to be approximated by Chebyshev Series.

2.1 For f(x):

The functions to be approximated are:

$$f(x) = \exp(x) \tag{13}$$

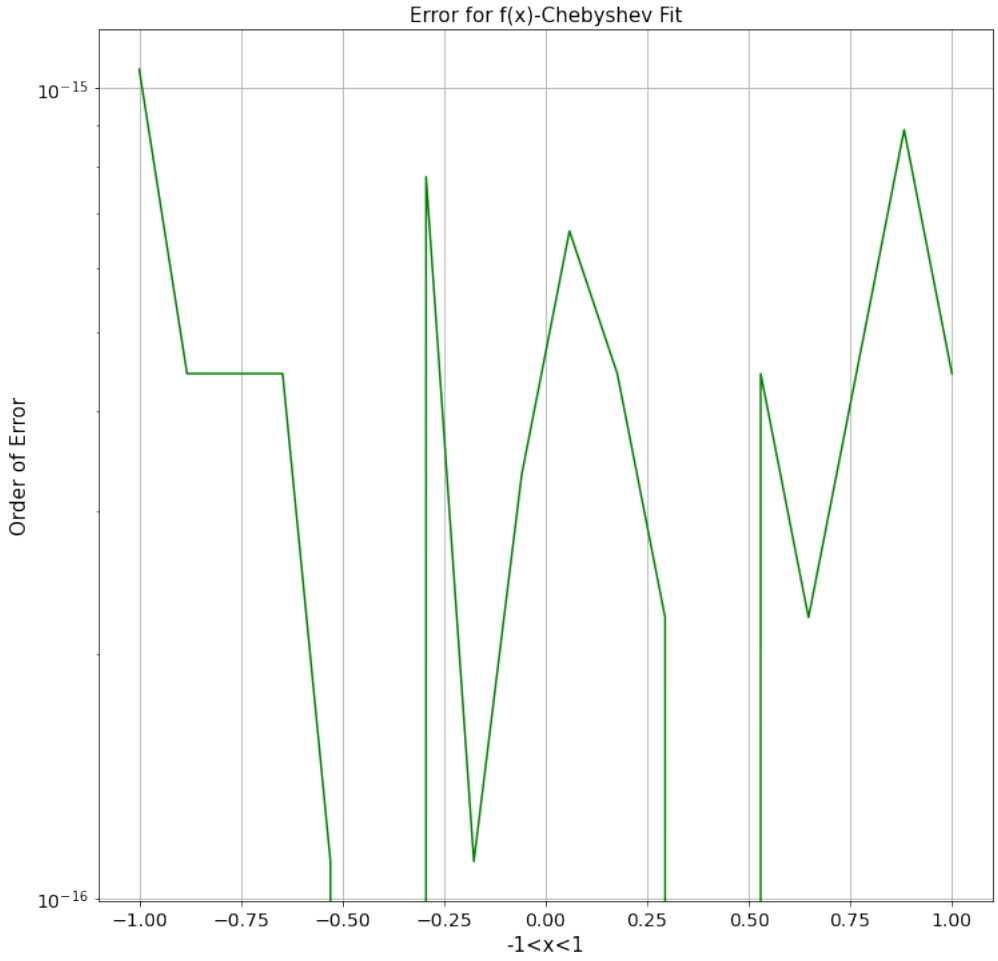

Figure 6

Here only 18 coefficients are taken.

2.2 For g(x):

The function to be approximated is:

$$g(x) = \frac{1}{x^2 + \delta^2} \tag{14}$$

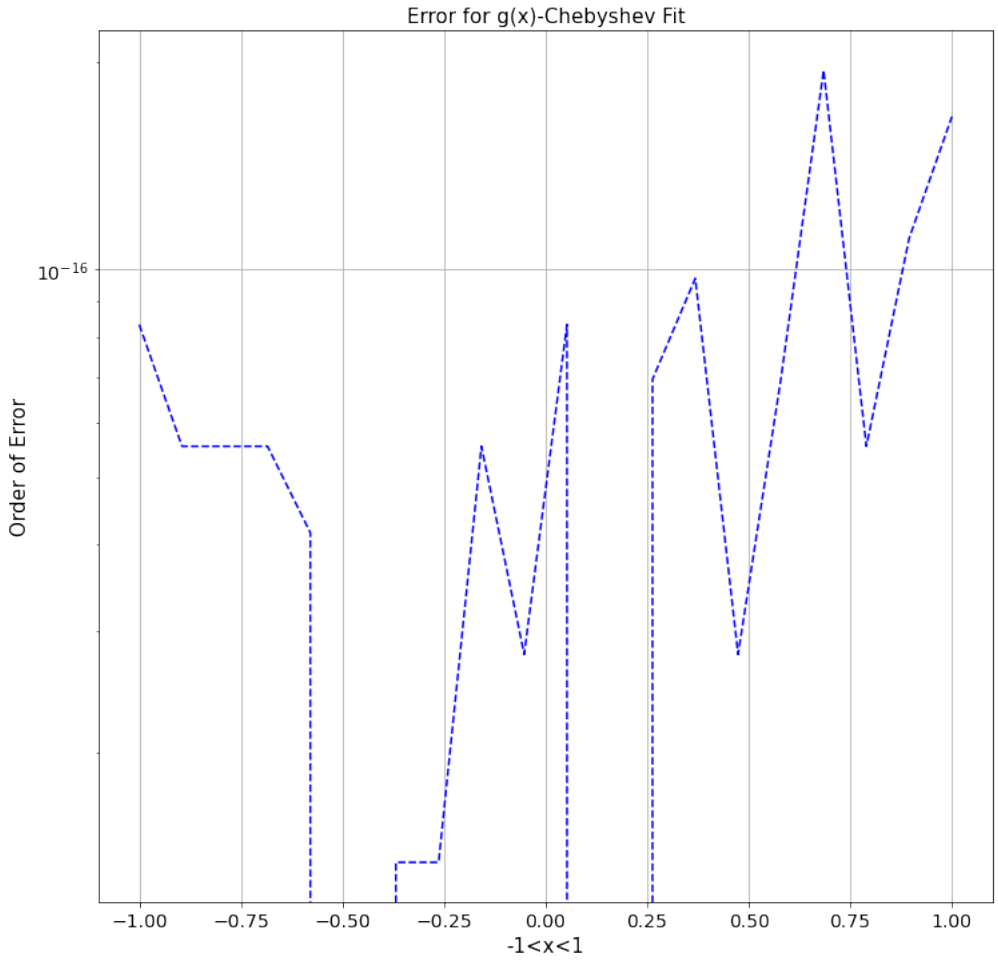

Figure 7

Here only 20 coefficients are taken.

2.3 For h(x)

The function to be approximated is:

$$h(x) = \frac{1}{\cos^2(\pi x/2) + \delta^2} \tag{15}$$

From Figure 5 it's evident that as the function is oscillatory its harder to contain it using Chebyshev, thus it takes around 30 coefficients to bring it down to machine precision.

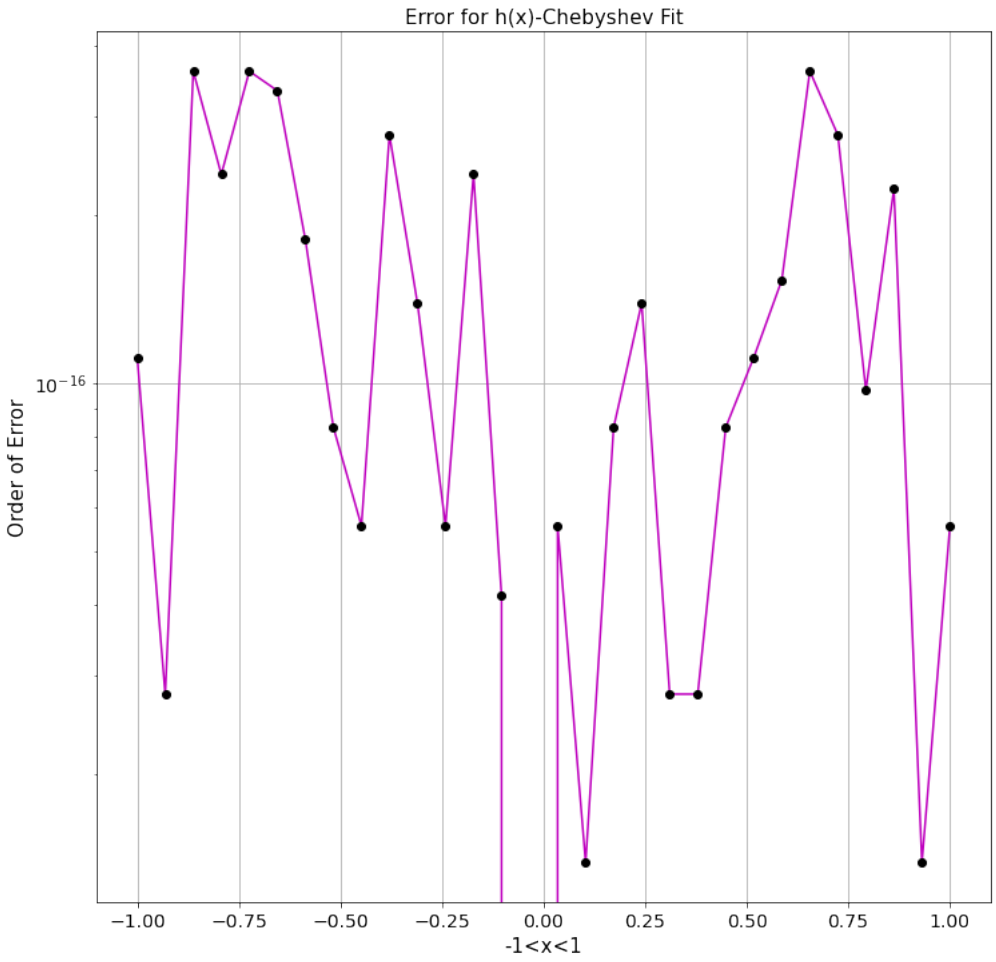

Figure 8

2.4 For u(x)

The function to be approximated is:

$$u(x) = \exp(-|x|) \tag{16}$$

Coefficients

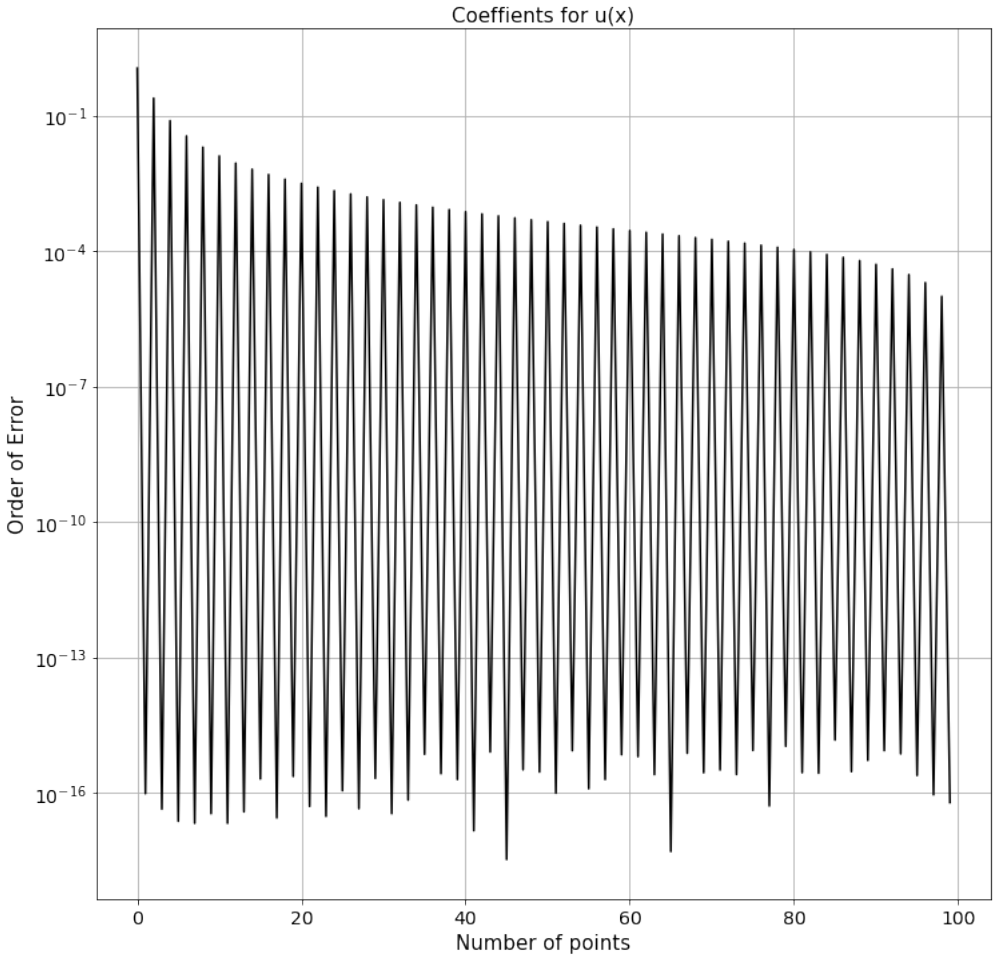

Figure 9

From Figure 9, its evident that the coefficients are oscillatory and doesn't fall below much from 10^{-4}, thus I took just 50 coefficients to approximate.

The corresponding graph for error with those coefficients is: The whole range

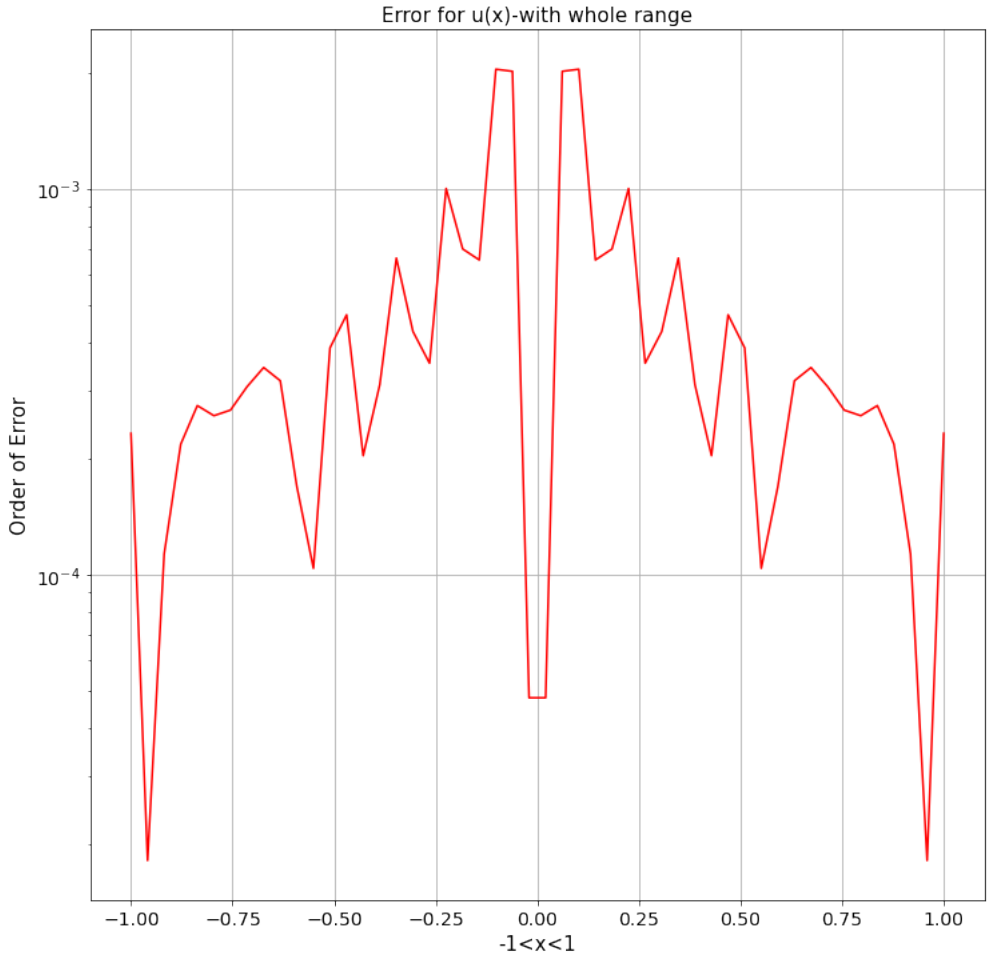

Figure 10

for $-1 < x < 1$ the function g(x) doesn't give good results, so the range is broken as $-1 < x < 0$ and $0 < x < 1$ and then the coefficients are calculated.

The error when ranges are broken down,

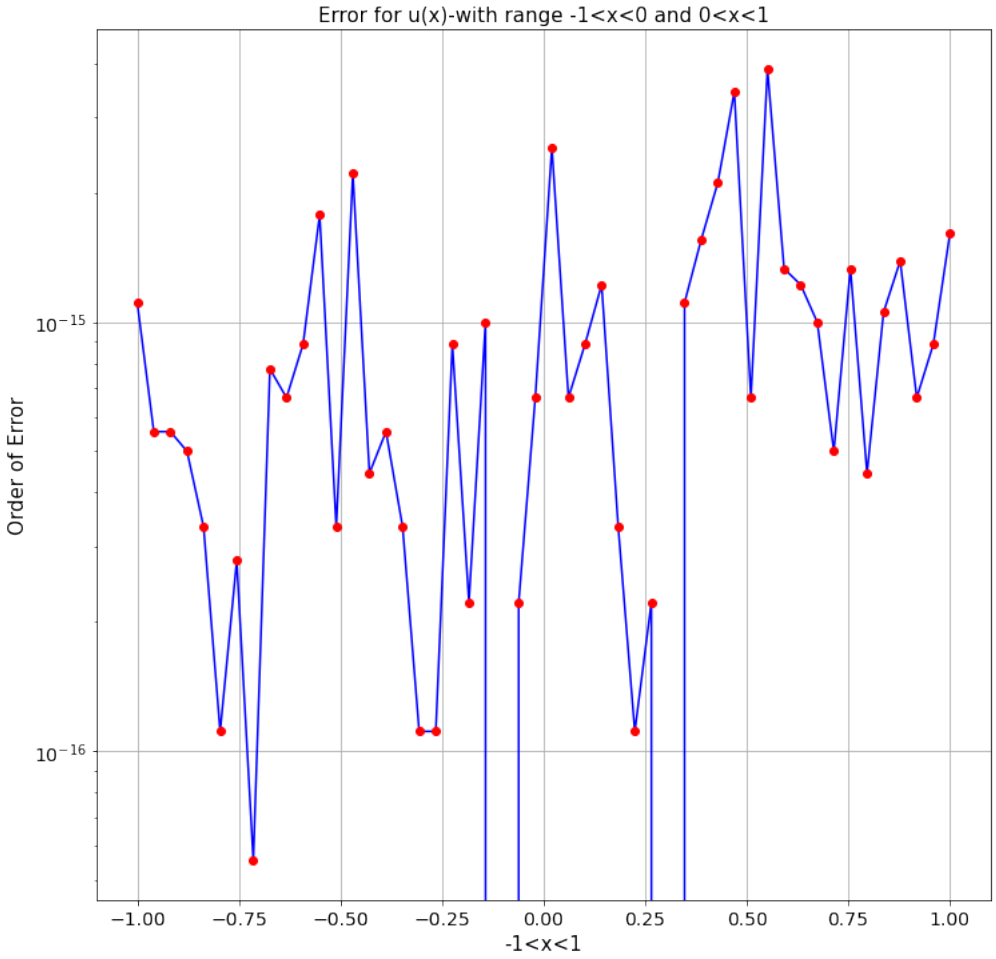

Figure 11

2.5 For v(x)

The function to be approximated is:

$$v(x) = \sqrt{x + 1.1} \tag{17}$$

The coefficients for v(x) falls slowly compared to that to other function's coefficients as shown in Figure 5.
With just 20 coefficients the error is around order 6,

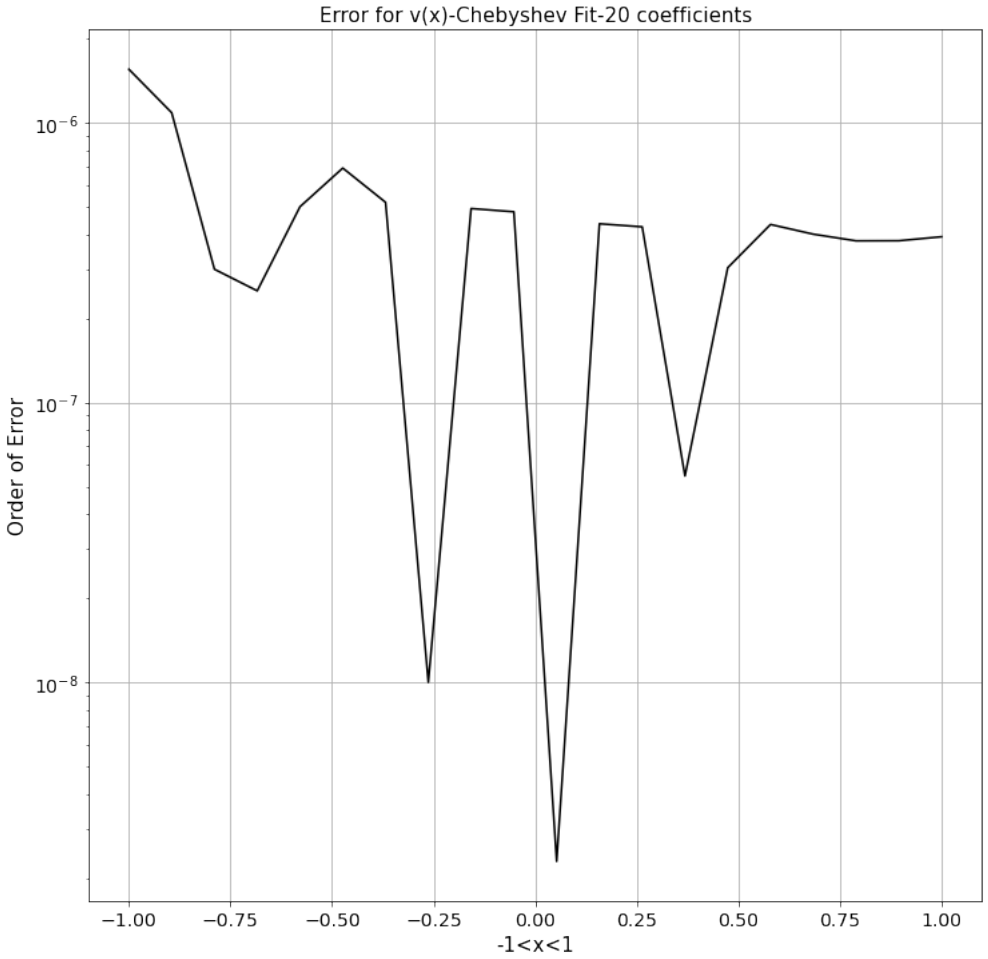

Figure 12

Thus the number of coefficients have to be more to bring it down to machine precision.When the number of points taken are 60, the below error graph is obtained.

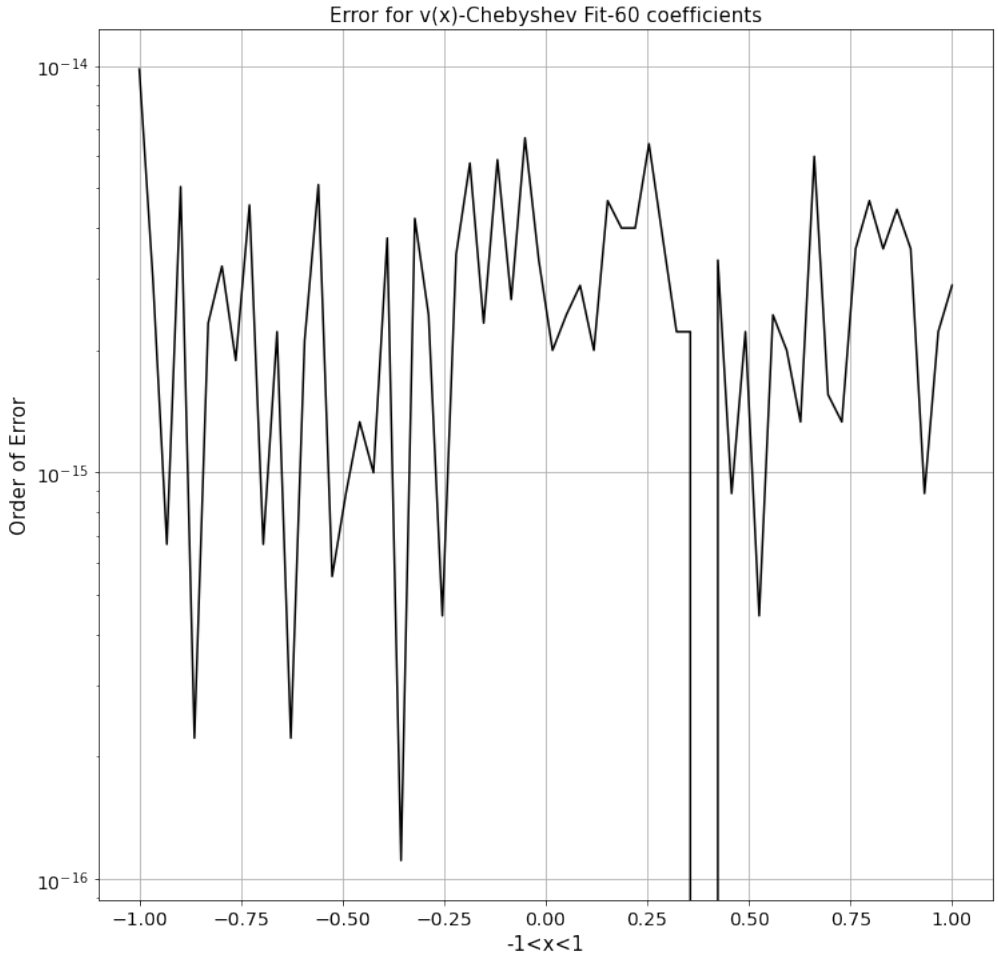

Figure 13

Overall, v(x) doesn't perform well with Chebyshev.

2.6 Coefficients of Function, g(x) and h(x) with different deltas

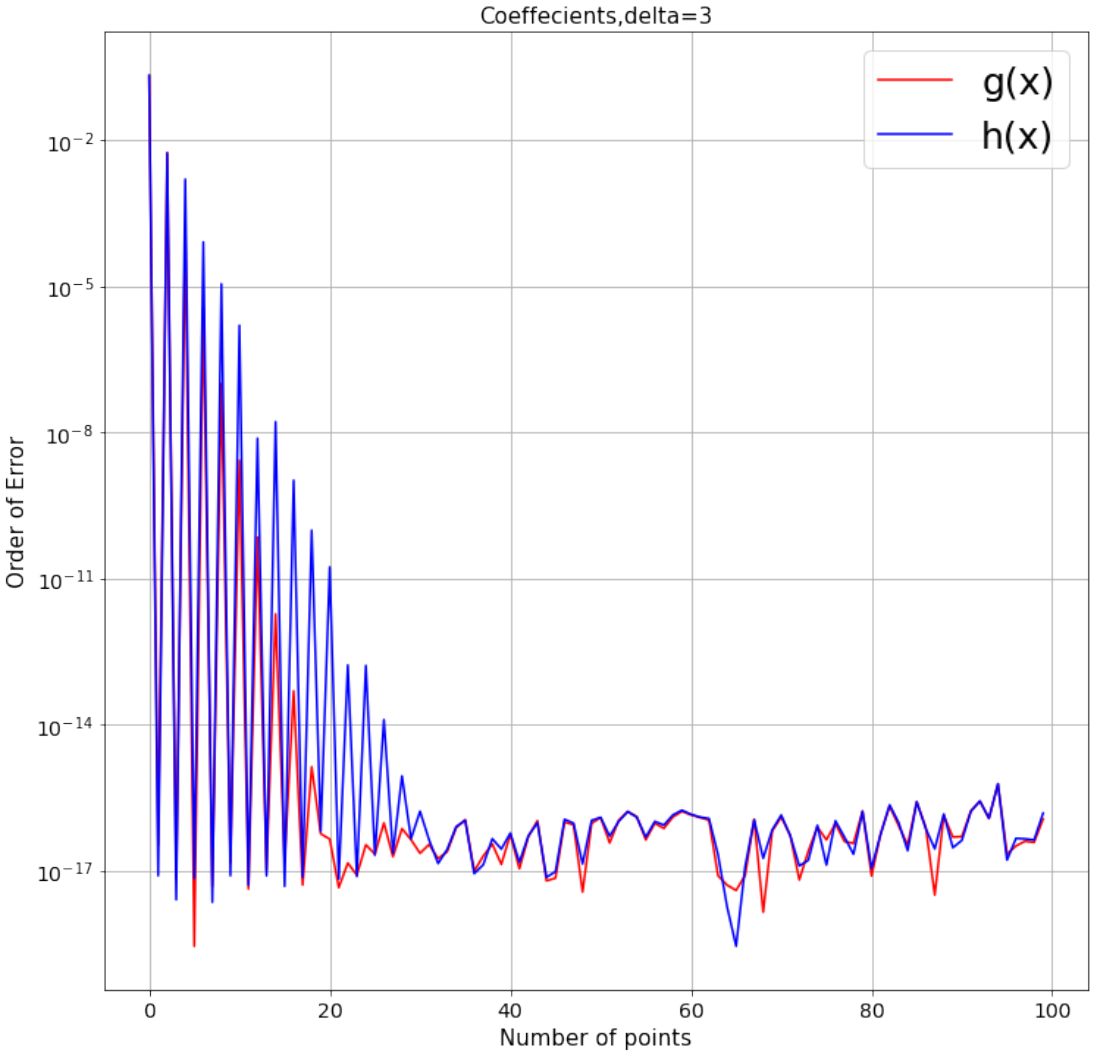

Figure 14: Delta=3, Cutoff= 30 pts

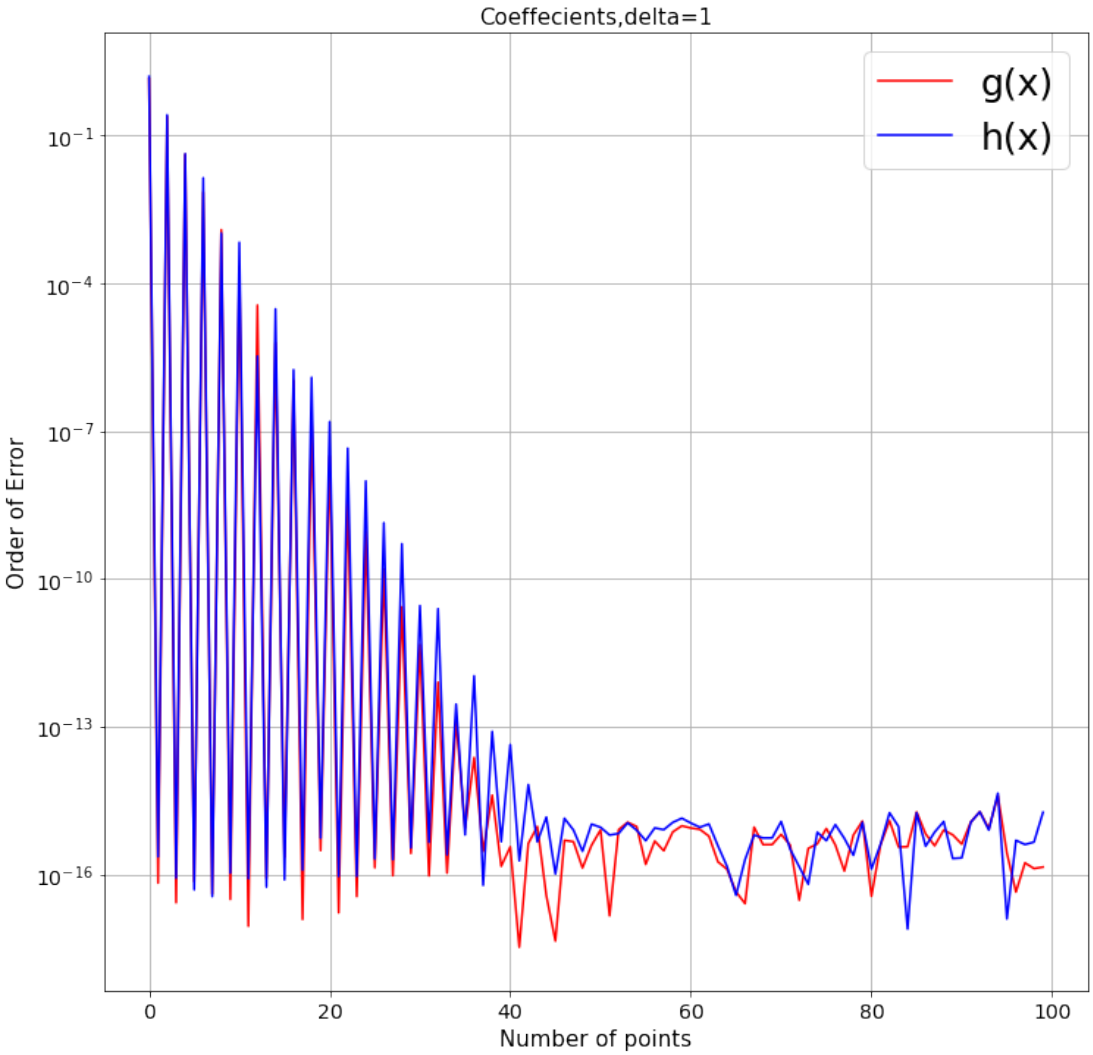

Figure 15: Delta=1, Cutoff= 50 pts

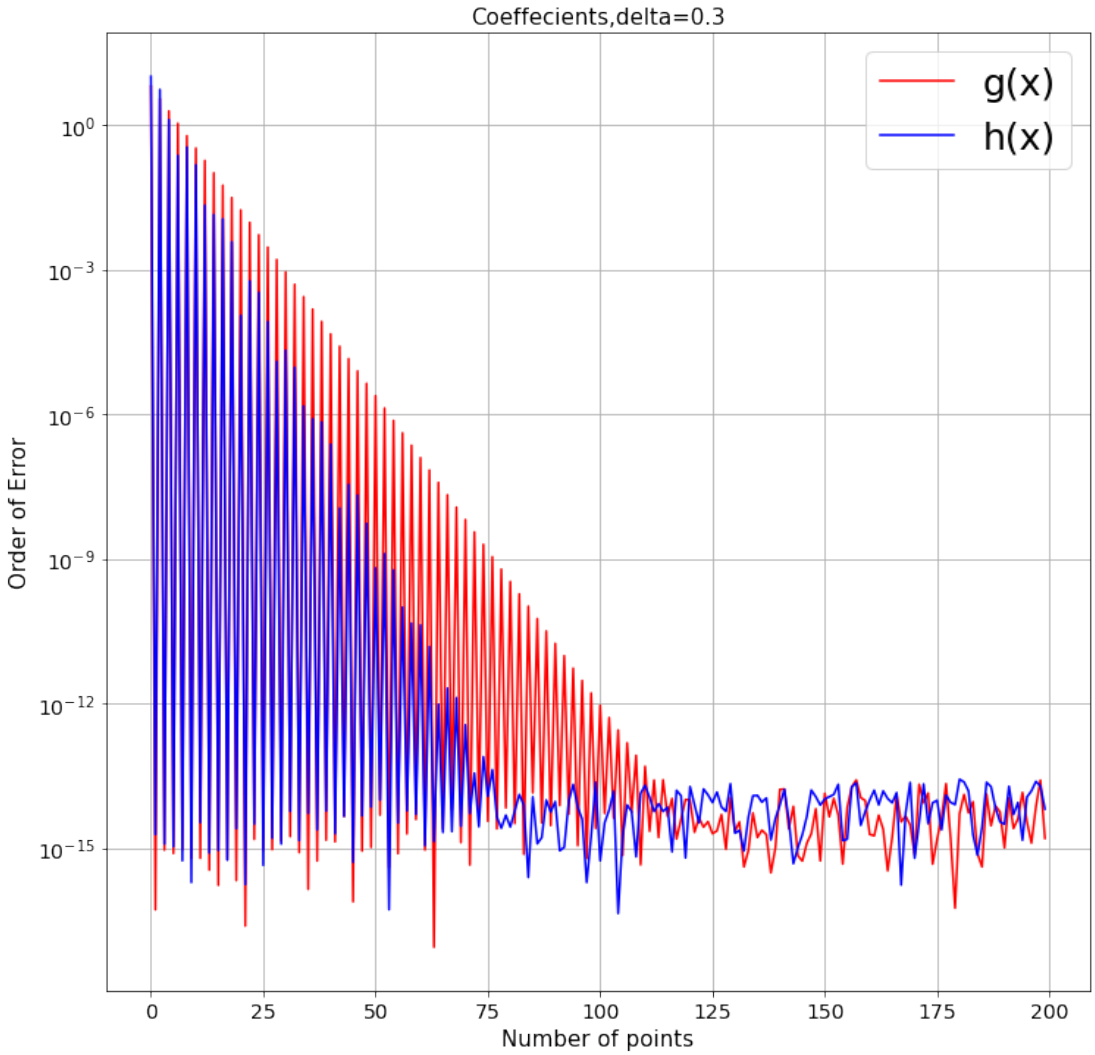

Figure 16: Delta=0.3, Cutoff= 125 pts

We can see that with decreasing deltas, the number of approximation terms required are increasing, so larger the delta the faster the approximation converges to machine precision.

2.7 Fourier coefficients

For f(x), the graph is,

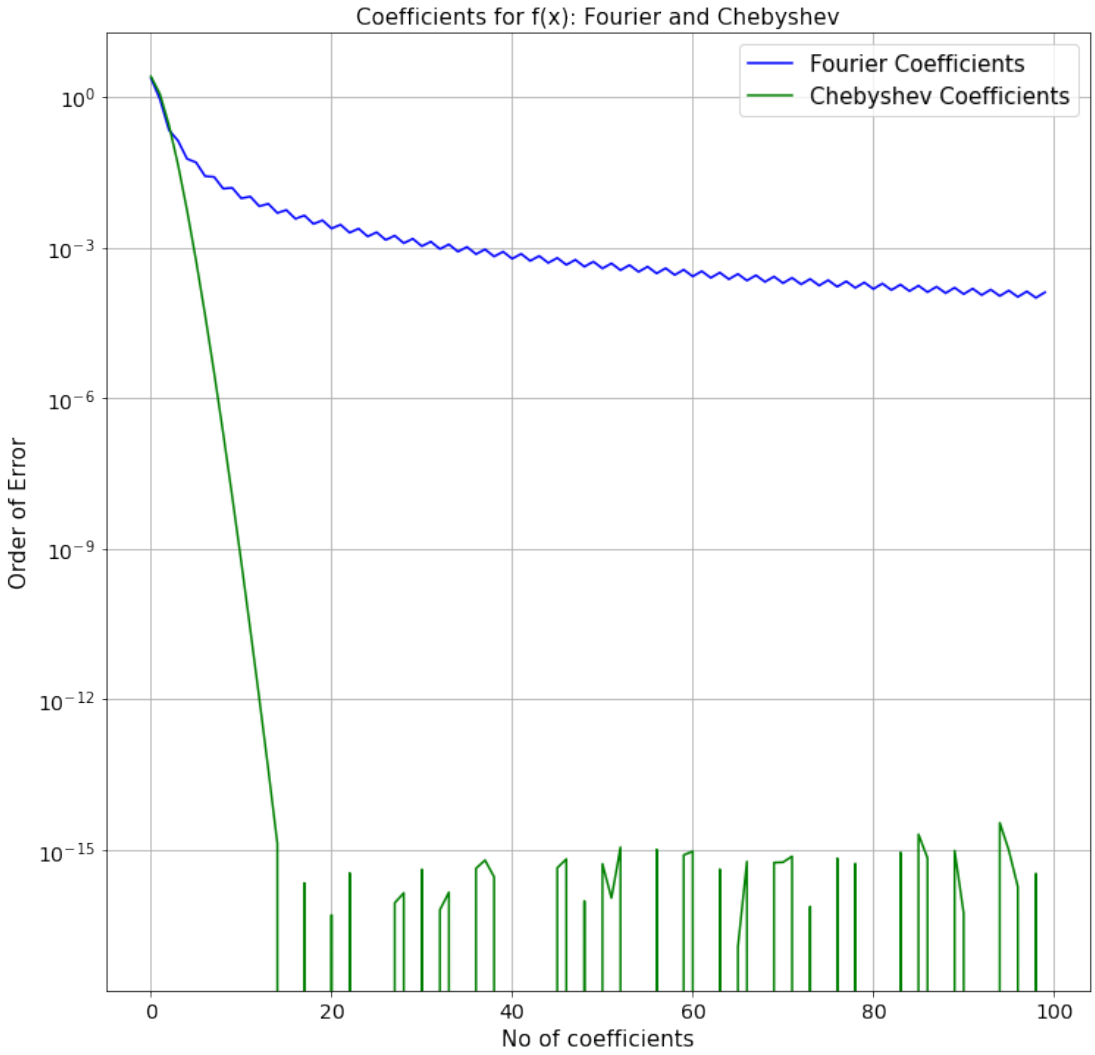

Figure 17

For g(x), the graph is,

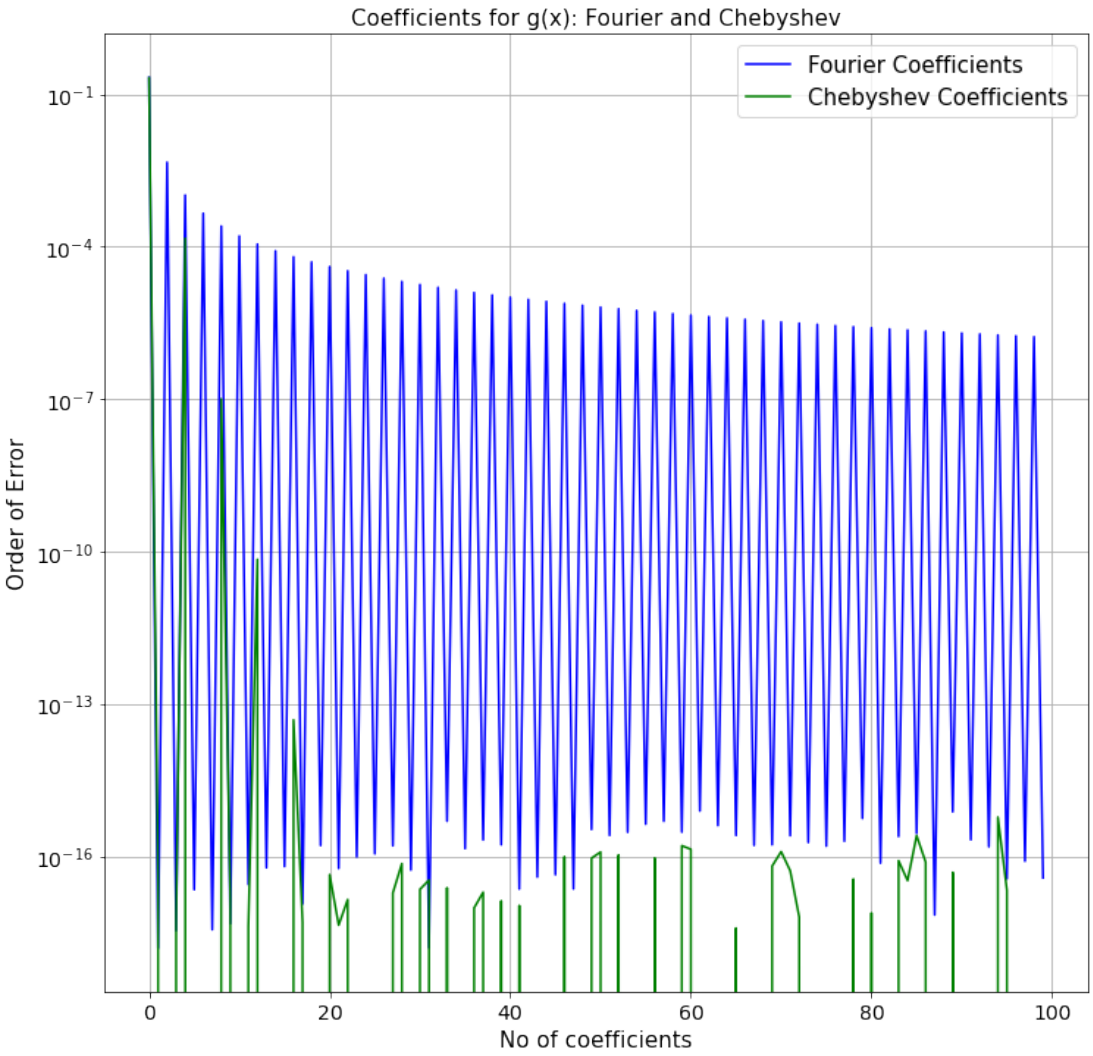

Figure 18

For h(x), the graph is,

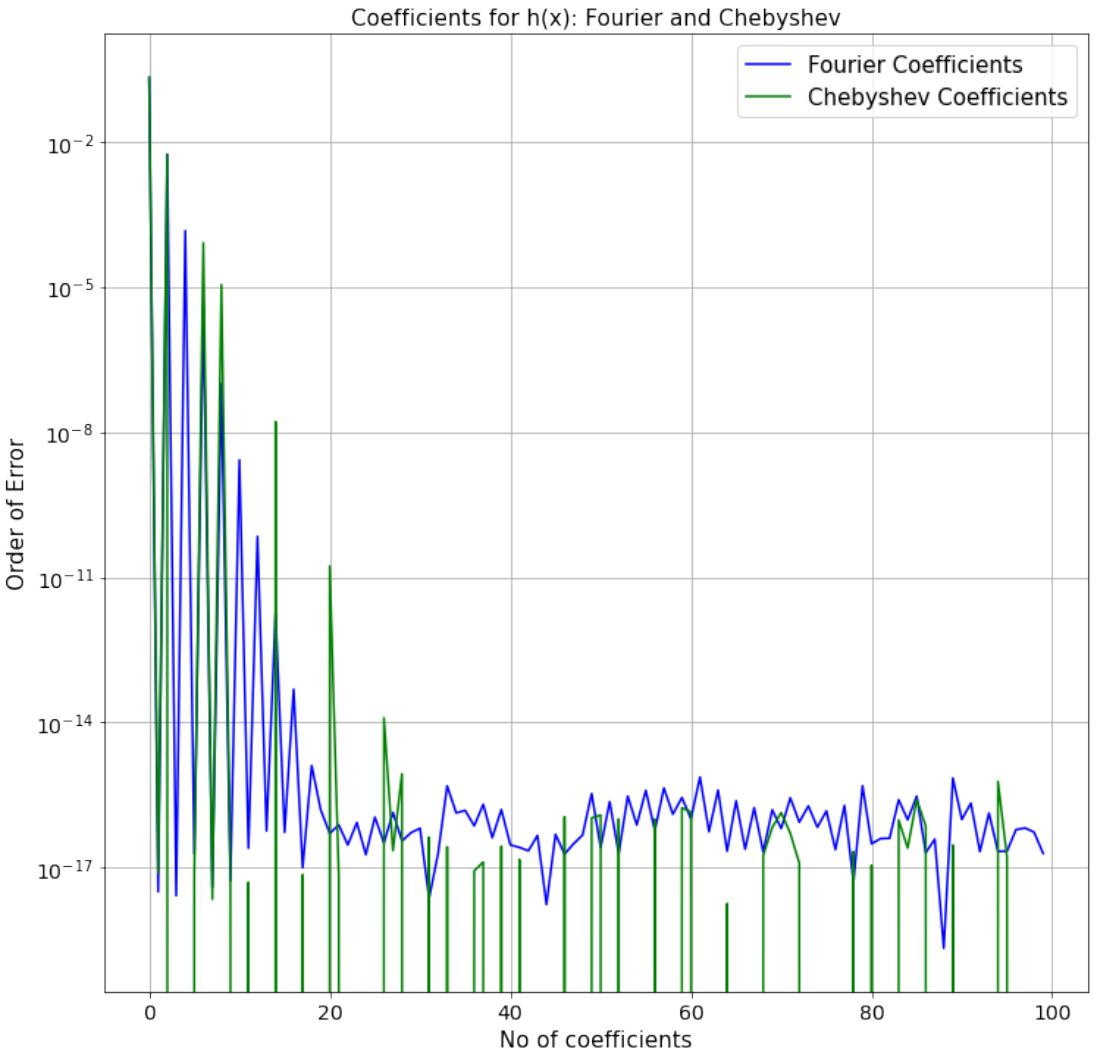

Figure 19

For u(x), the graph is,

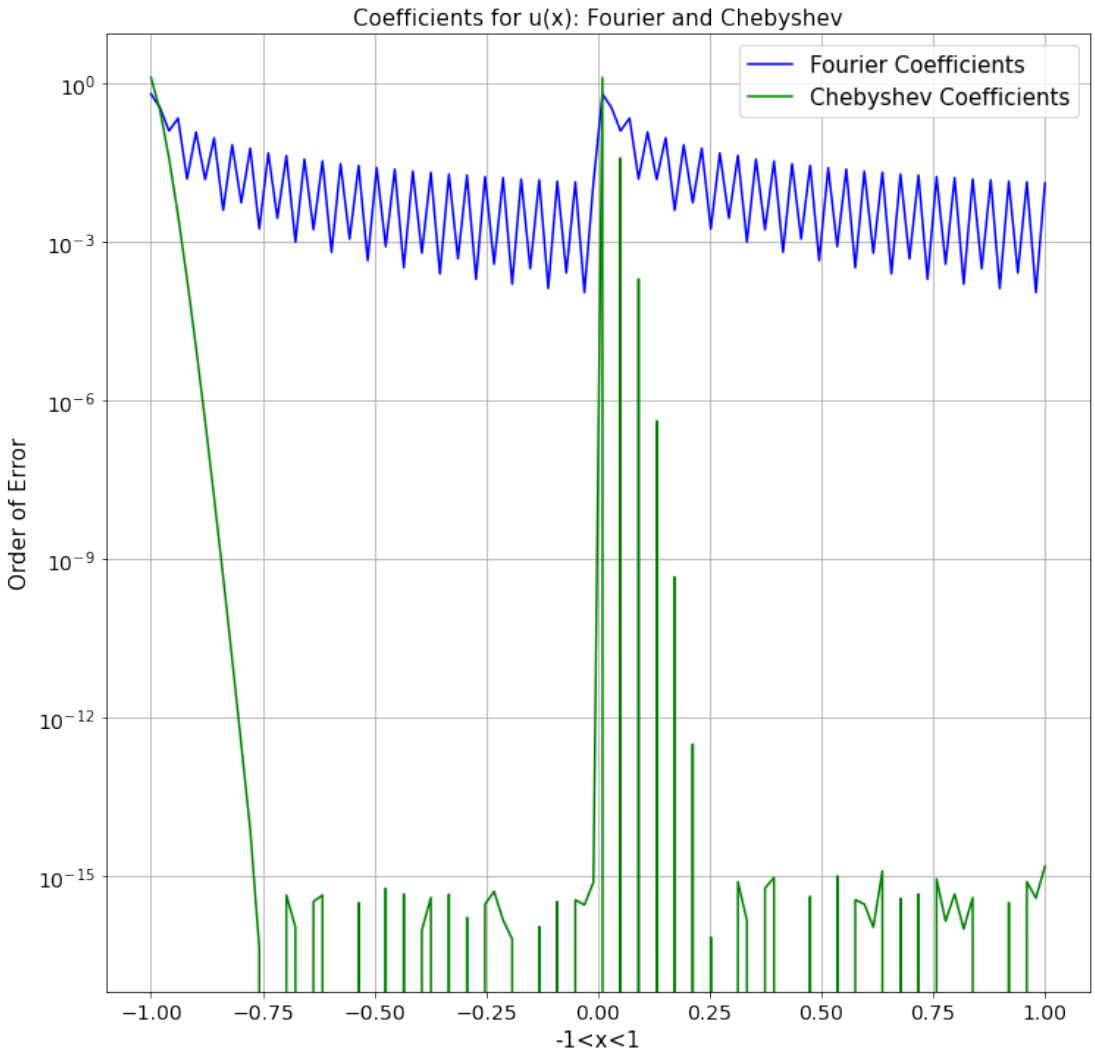

Figure 20

For v(x), the graph is,

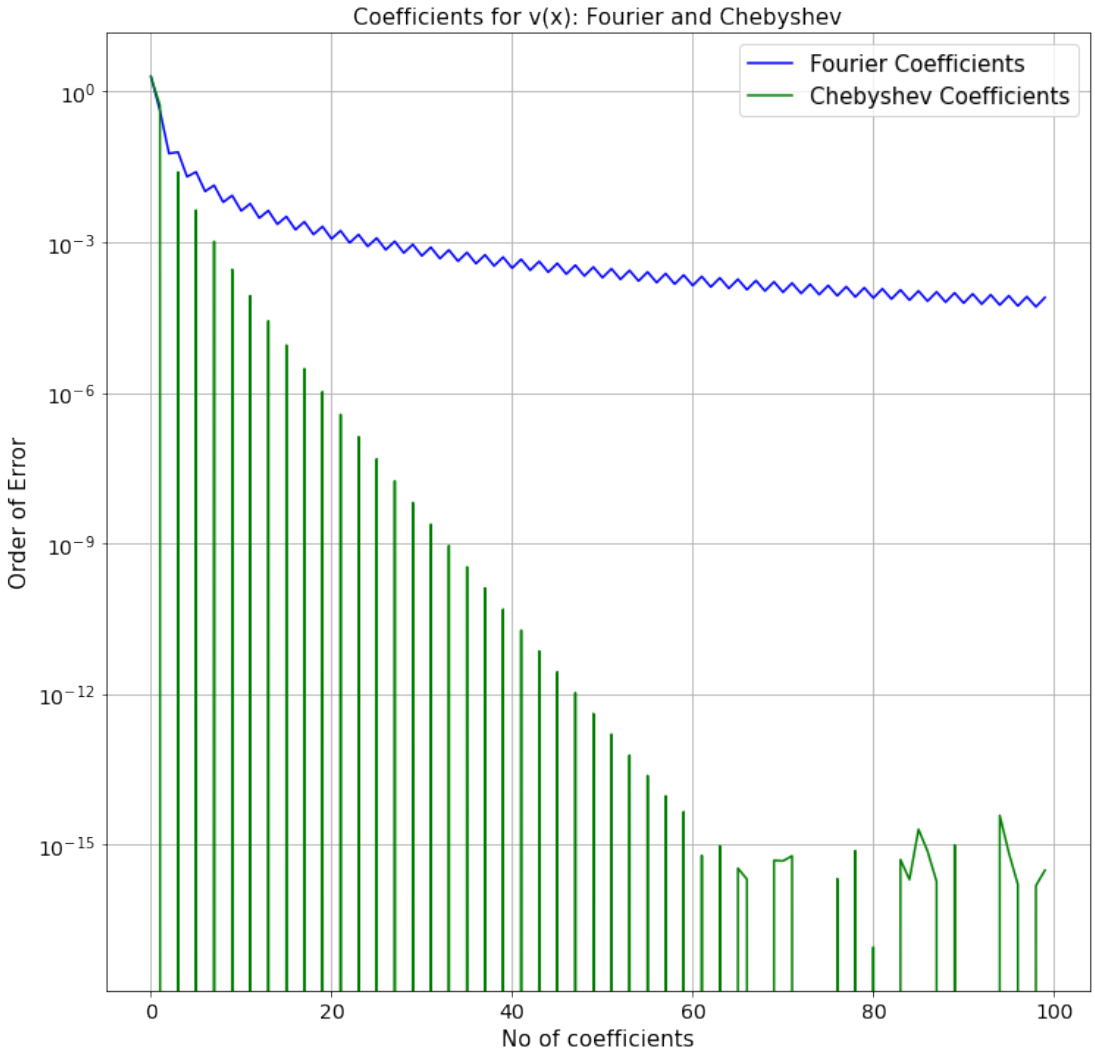

Figure 21

Thus Fourier approximations are not worth it compared to chebyshev, however we can use Fourier for a periodic function, like h(x) in our case.

2.8 Comparative analysis of error from Fourier and Chebyshev, using Clenshaw

The Fourier approximation gives very bad error, as evident from the plots below. For f(x),

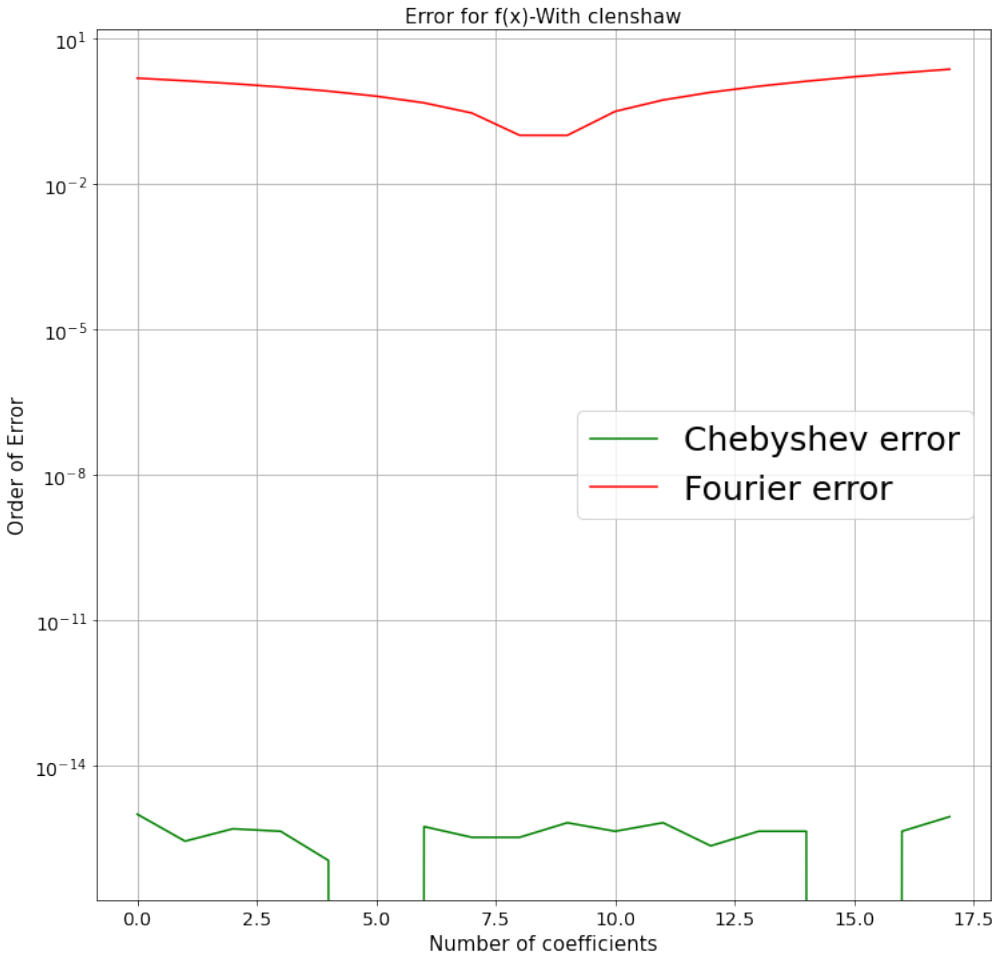

Figure 22

For g(x),

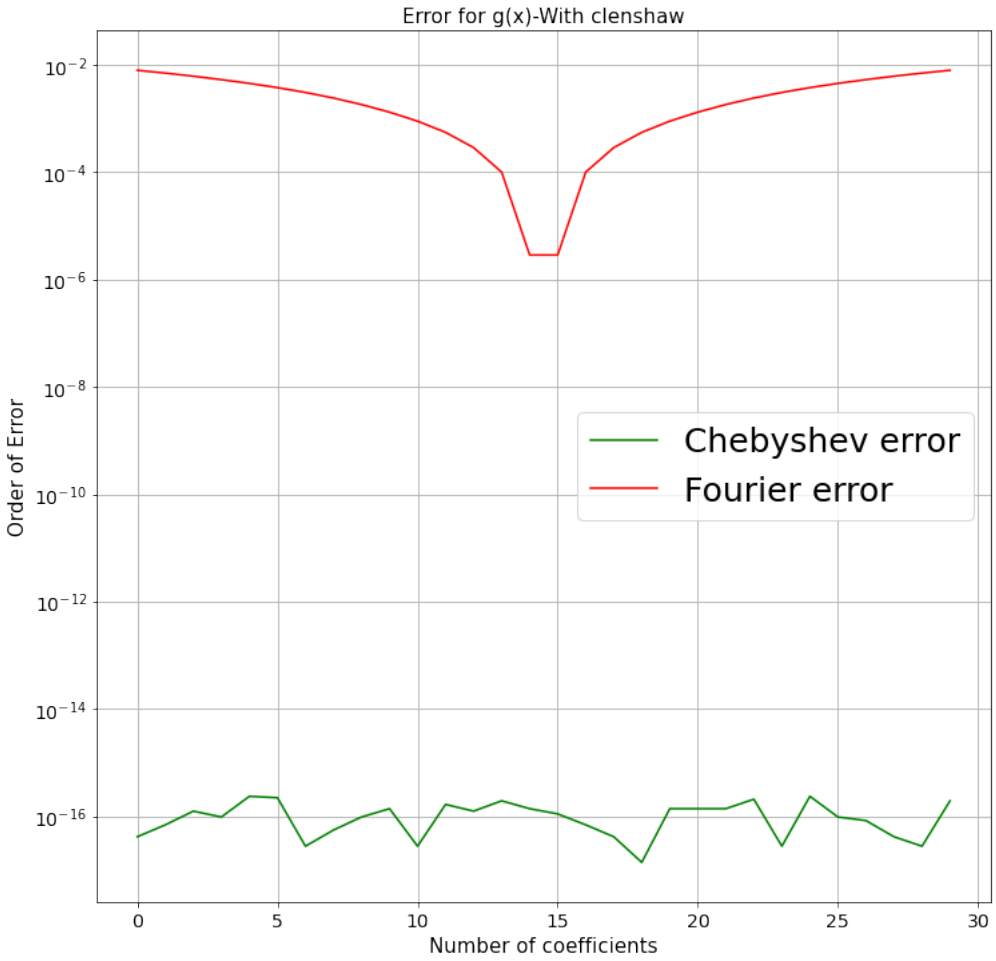

Figure 23

For h(x),

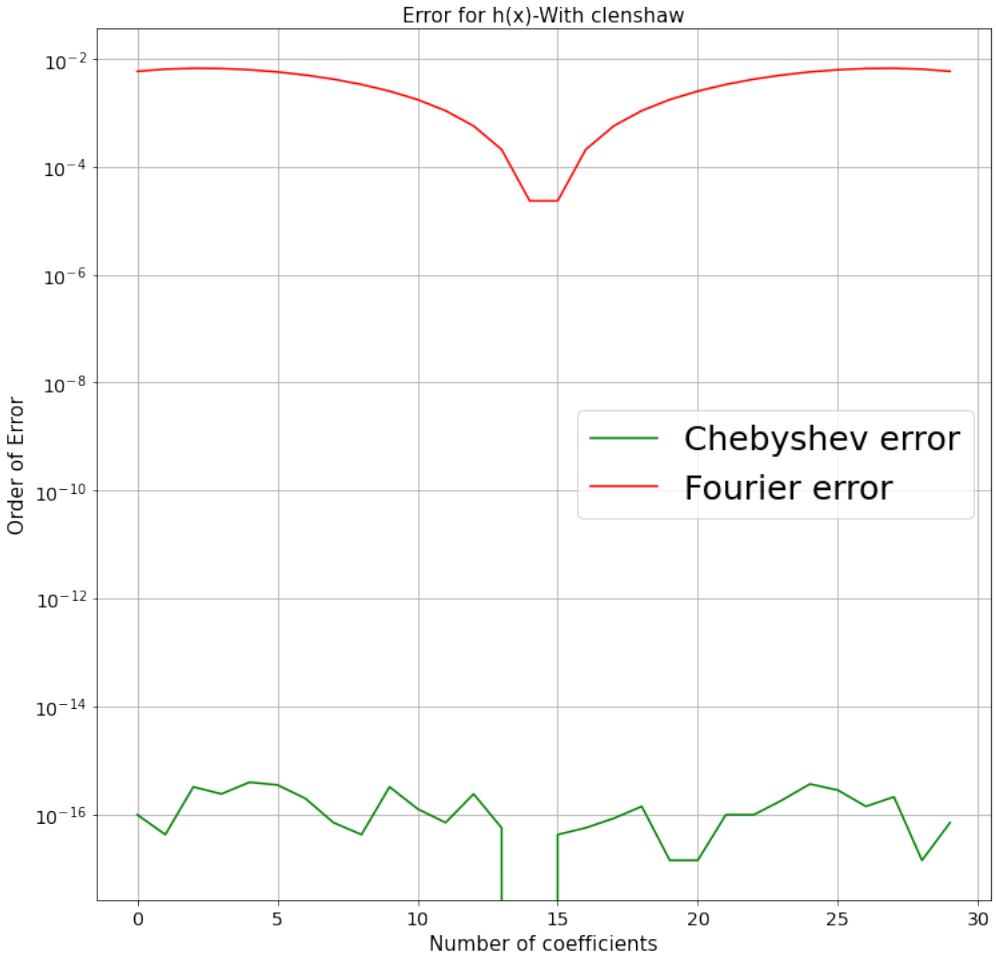

Figure 24

For u(x),

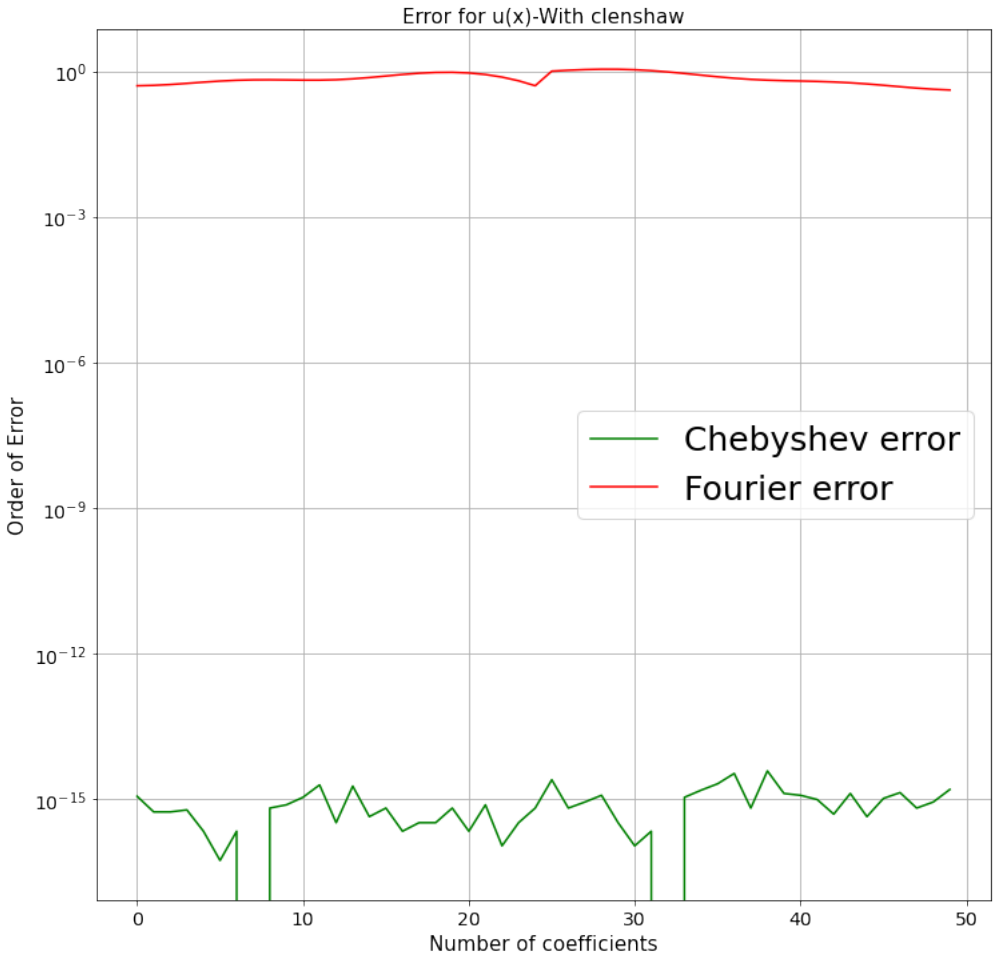

Figure 25

Here the range of x has been divided first, with $-1 < x < 0$ and $0 < x < 1$ and coefficients are calculated accordingly.

For v(x),

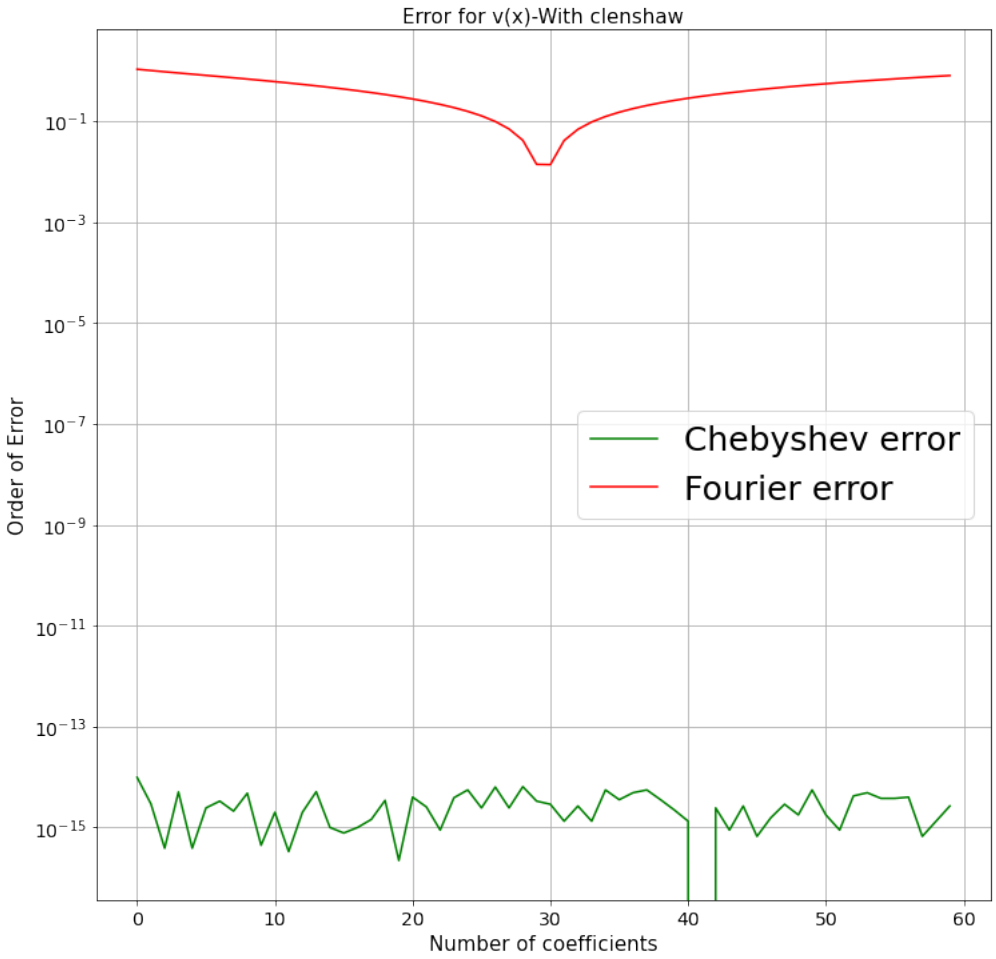

Figure 26

This clearly shows fourier approximations aren't worth it.