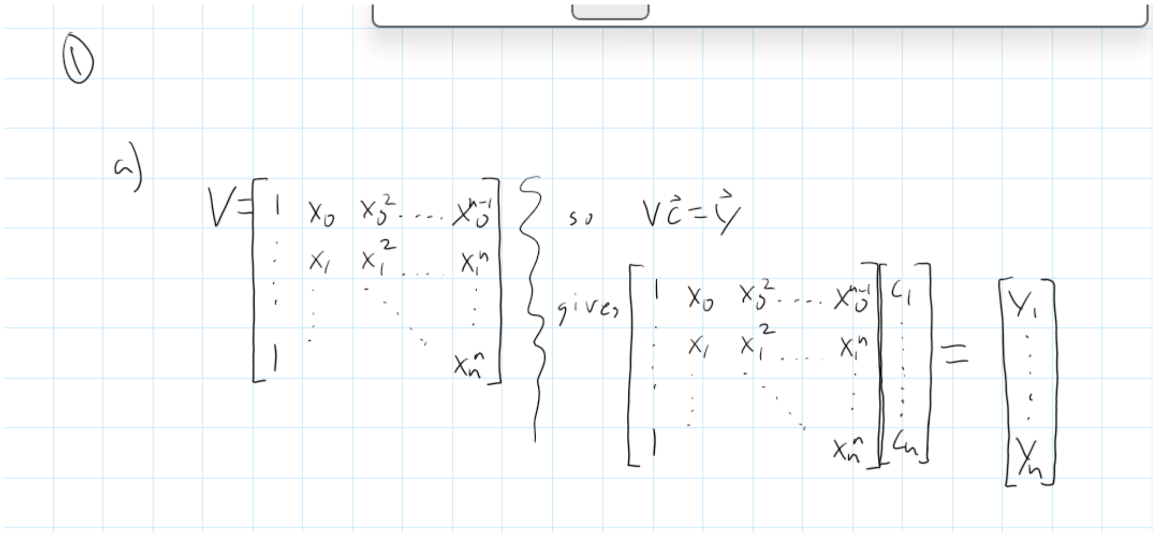
Homework 7

Problem 1

Vandermonde function:

a)

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import interp as lag
```



① a)

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ \vdots & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & & & & x_n^n \end{bmatrix}$$

so $V\vec{c} = \vec{y}$ gives

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ \vdots & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & & & & x_n^n \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix}$$

In [2]:
```python
def vander(V,y):
    sol = np.linalg.solve(V,y)
    return sol
```

b)

In [3]:
```python
N1=2
i1 = np.linspace(1,N1,N1)
x1= -1+(i1-1)*(2/(N1-1))
x1=x1.reshape((2,1))
y1= 1/(1+(10*x1)**2)
def V1(x,N):
    v=np.ones((N,1))
    #print(v)
    for n in range(1,N):
        #print(n)
        #print(x**n)

        v=np.hstack(([v,(x**n)]))
        #print(v)
    return v
def polyCoefficients(x, coeffs):

    order = len(coeffs)

    y = 0
    for i in range(order):
        y += coeffs[i]*x**i
    return y
```

Testing

In [4]:
```
print(x1**3)
```

```
[[-1.]
 [ 1.]]
```

In [5]:
```
print(V1(x1,N1))
print(y1)
print((x1**3))
```

```
[[ 1. -1.]
 [ 1.  1.]]
[[0.00990099]
 [0.00990099]]
[[-1.]
 [ 1.]]
```

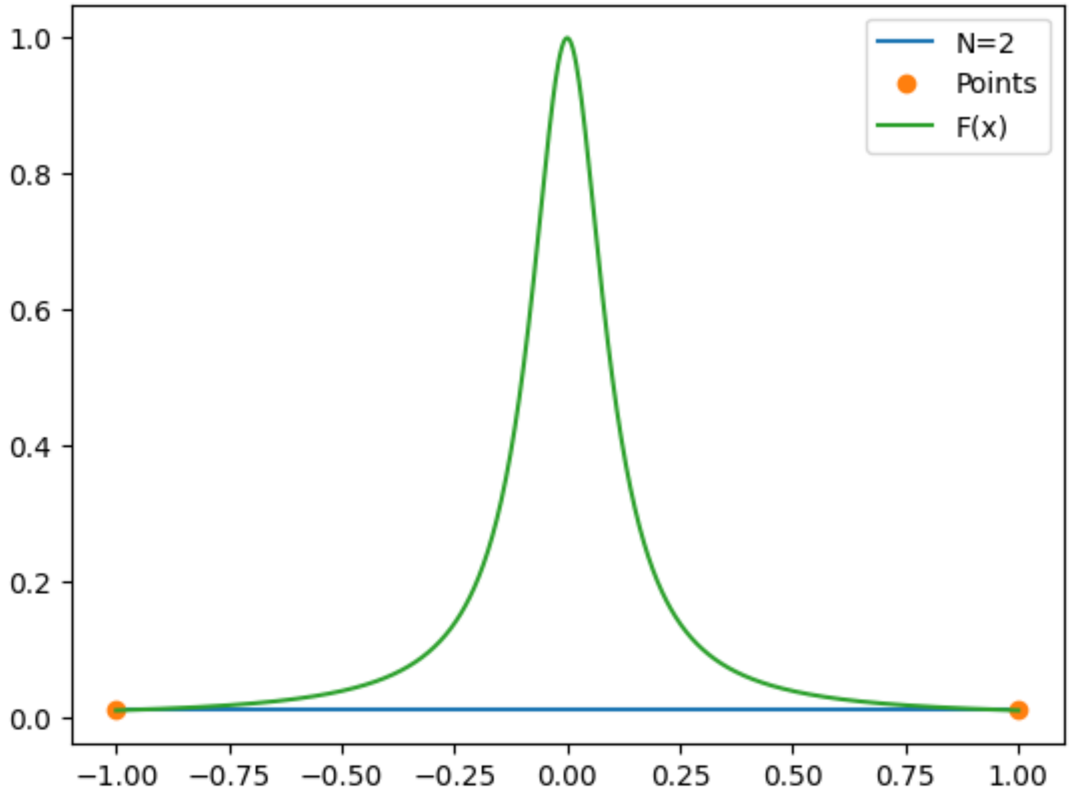
In [6]:
```
poly1 = vander(V1(x1,N1),y1)
```

In [7]:
```
print(poly1)
```

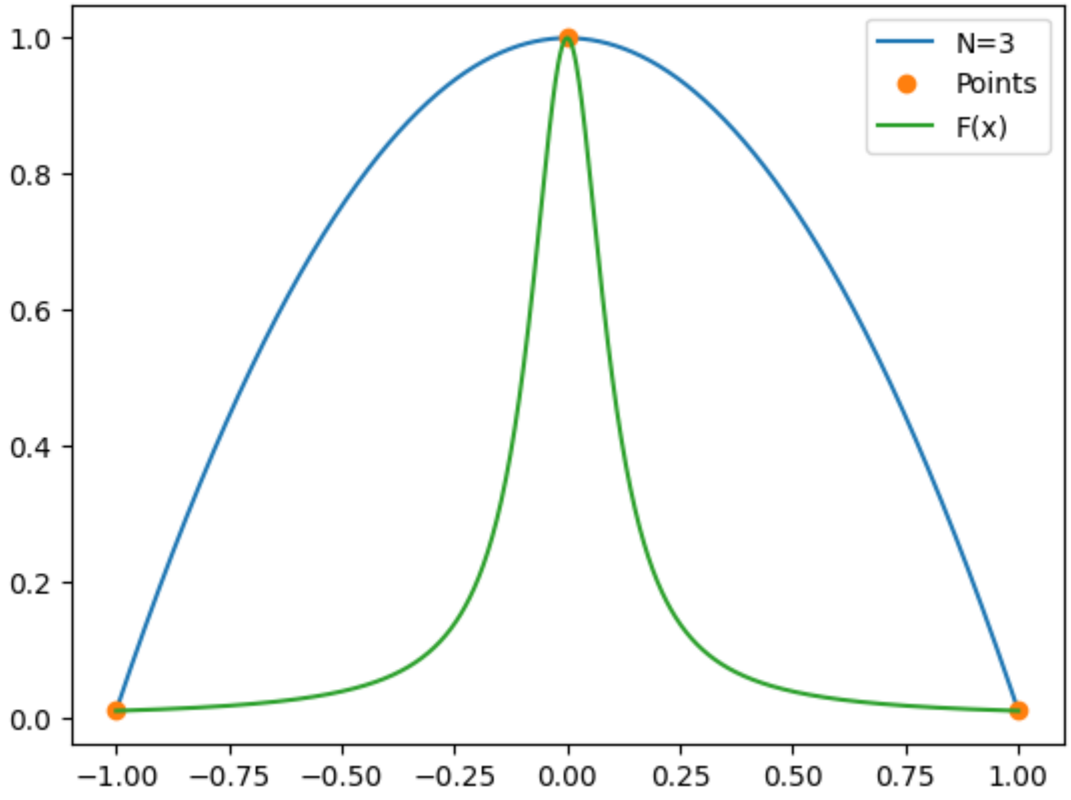
```
[[0.00990099]
 [0.        ]]
```

In [8]:
```
polyCoef = np.array(0)
poly=[0,0]
N2=2
xCont = np.linspace(-1,1,1001)
yCont = 1/(1+(10*xCont)**2)
while(max(poly)<100):
    i2 = np.linspace(1,N2,N2)
    x2= -1+(i2-1)*(2/(N2-1))
    x2=x2.reshape((N2,1))
    y2= 1/(1+(10*x2)**2)

    polyCoef = vander(V1(x2,N2),y2)
    poly = polyCoefficients(xCont,polyCoef)
    print("At N=",N2,"The max value is:",max(poly))
    plt.plot(xCont,poly, label = "N="+ str(N2))
    plt.plot(x2,y2,"o", label = "Points")
    plt.plot(xCont,yCont, label = "F(x)")
    plt.legend()
    plt.show()
    plt.close()
    N2+=1
```

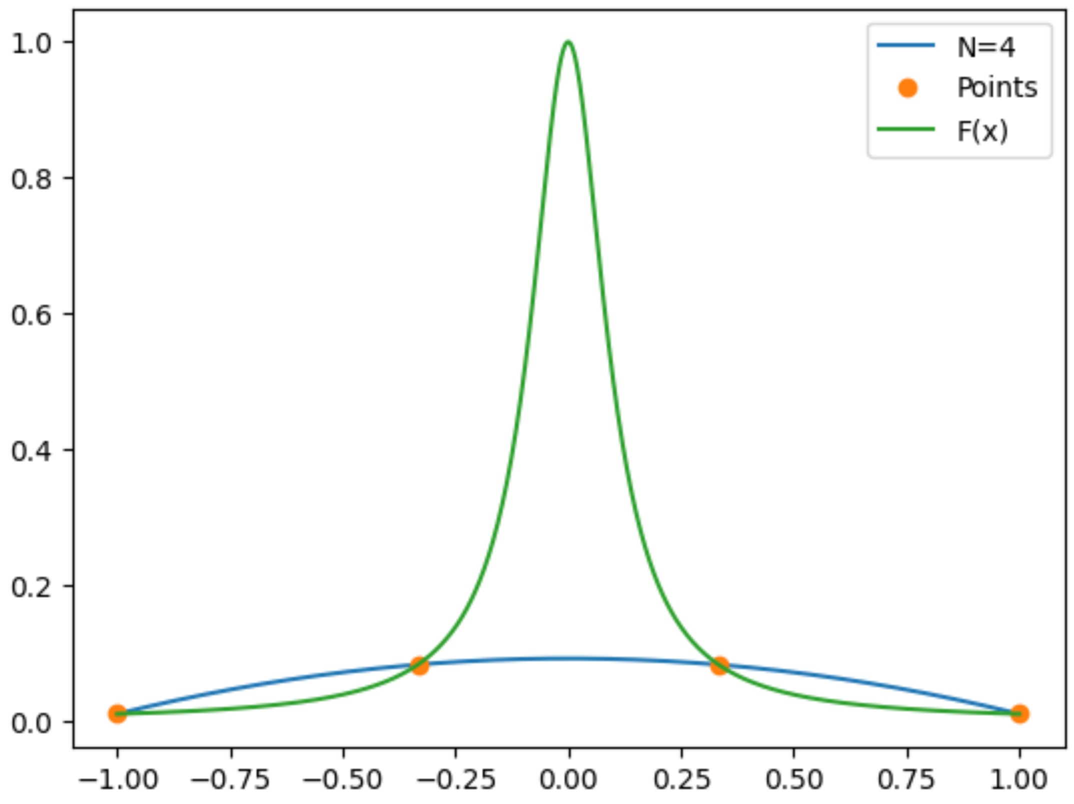
```
At N= 2 The max value is: 0.009900990099009901
```



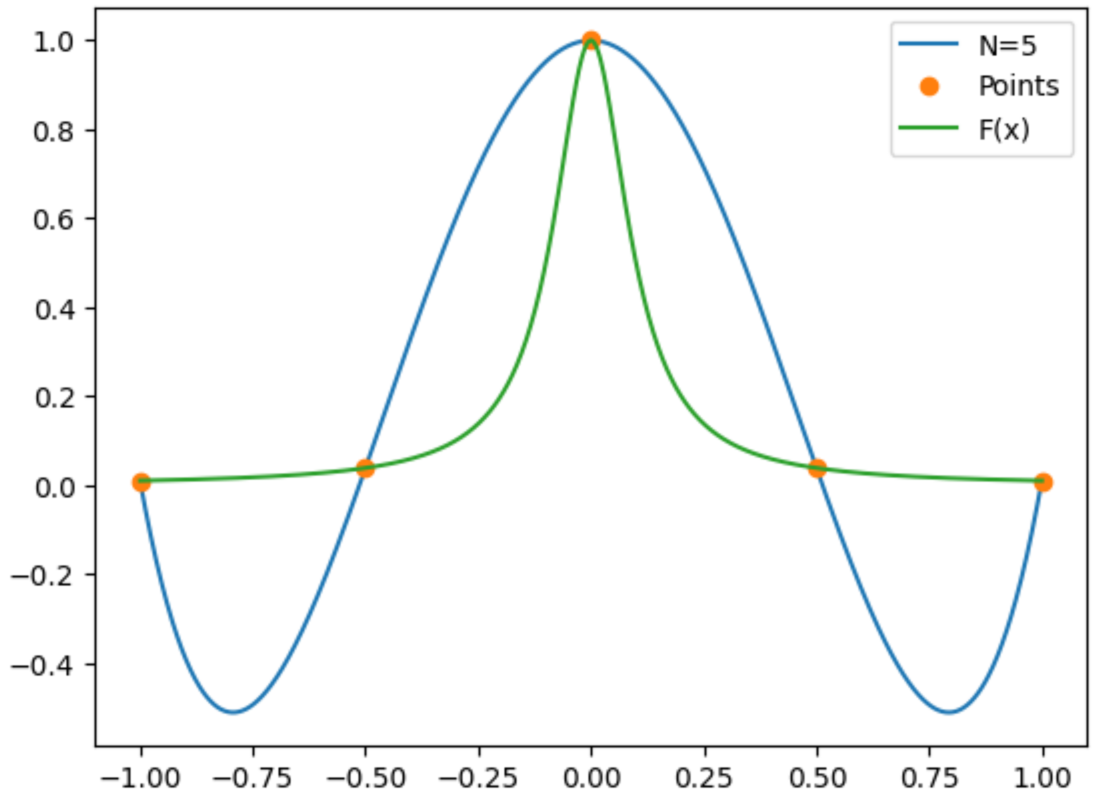
```
At N= 3 The max value is: 1.0
```



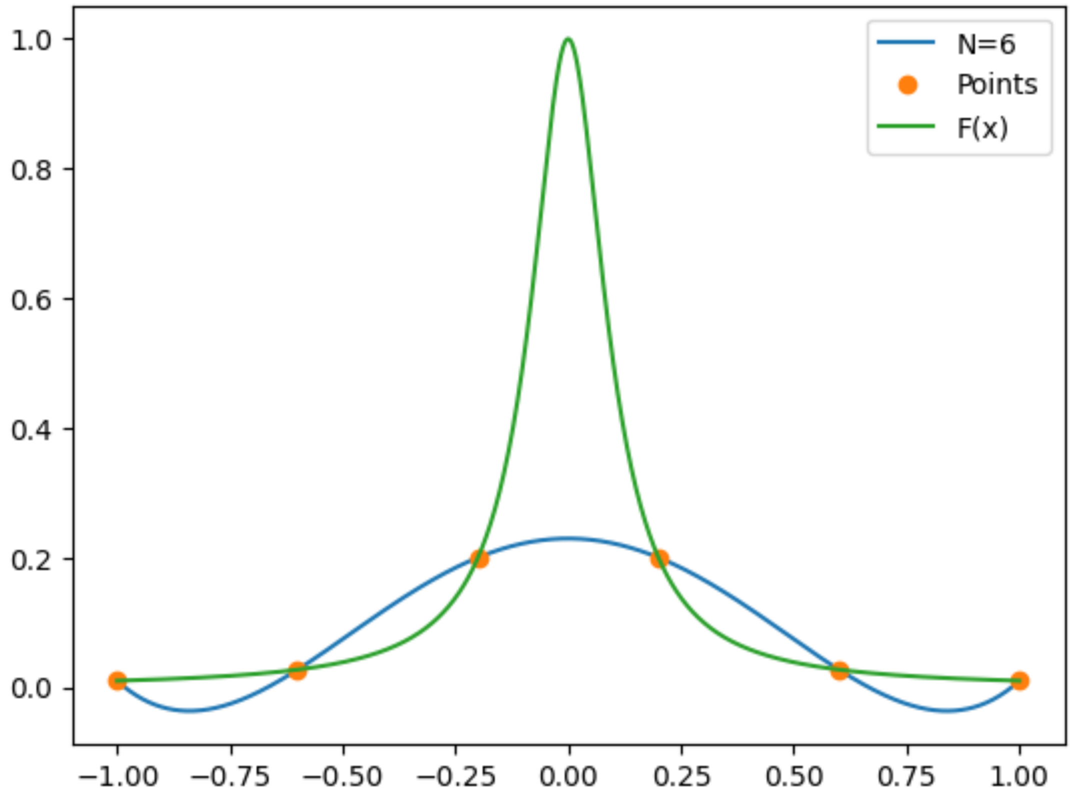
```
At N= 4 The max value is: 0.09165228449450451
```



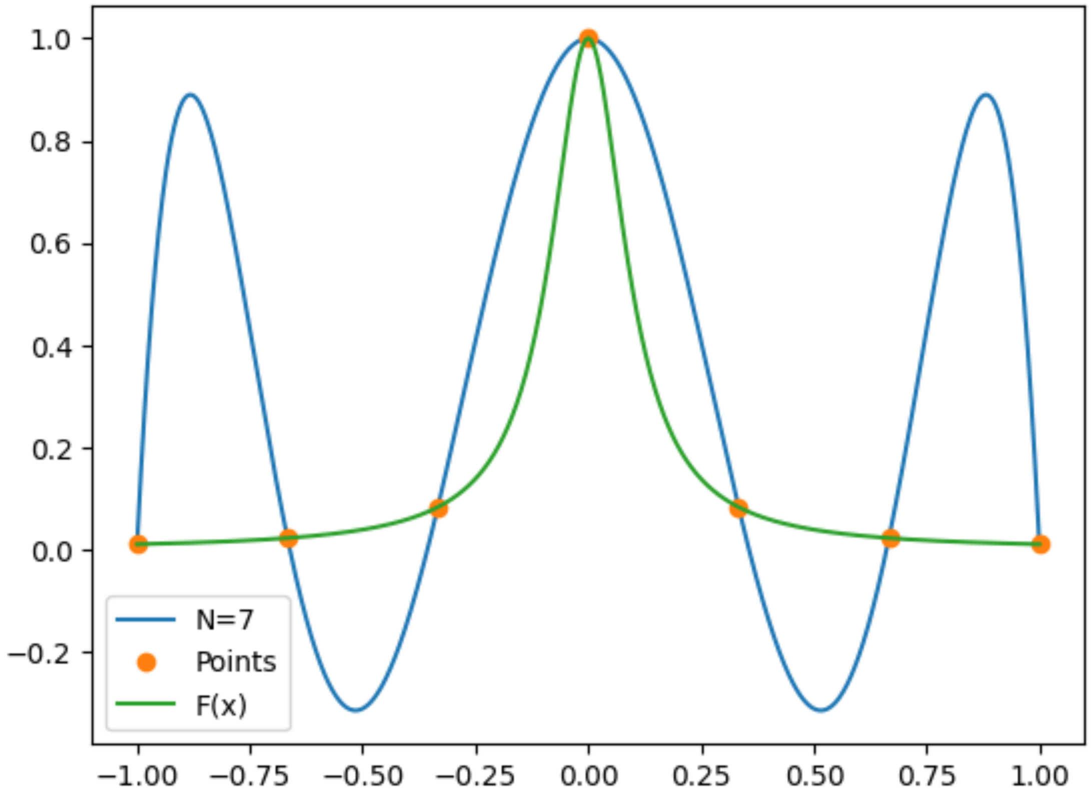
```
At N= 5 The max value is: 1.0000000000000004
```



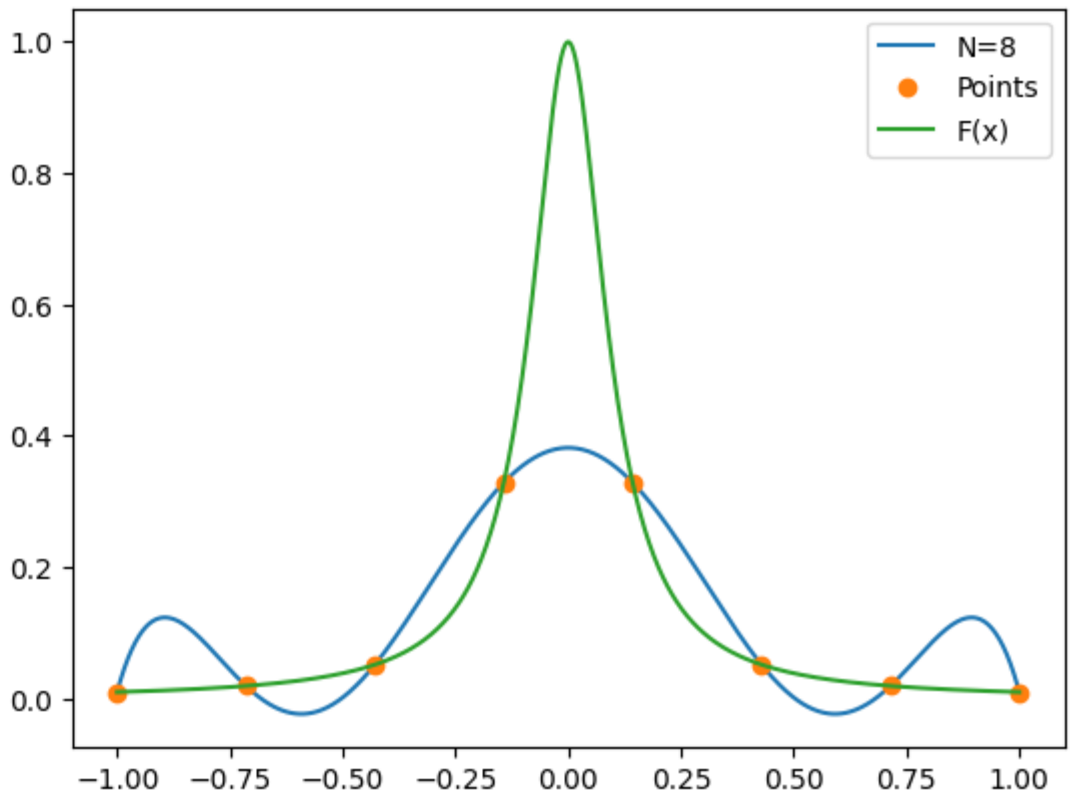
```
At N= 6 The max value is: 0.22932833823922913
```



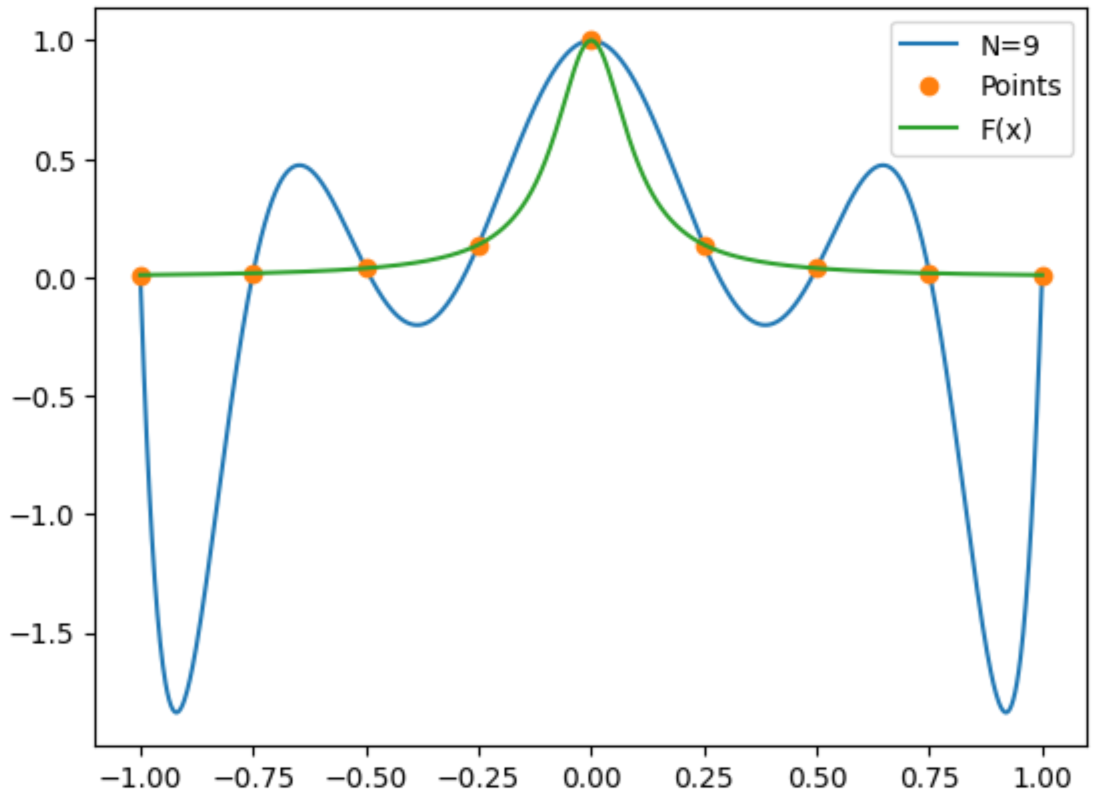
```
At N= 7 The max value is: 0.9999999999999996
```



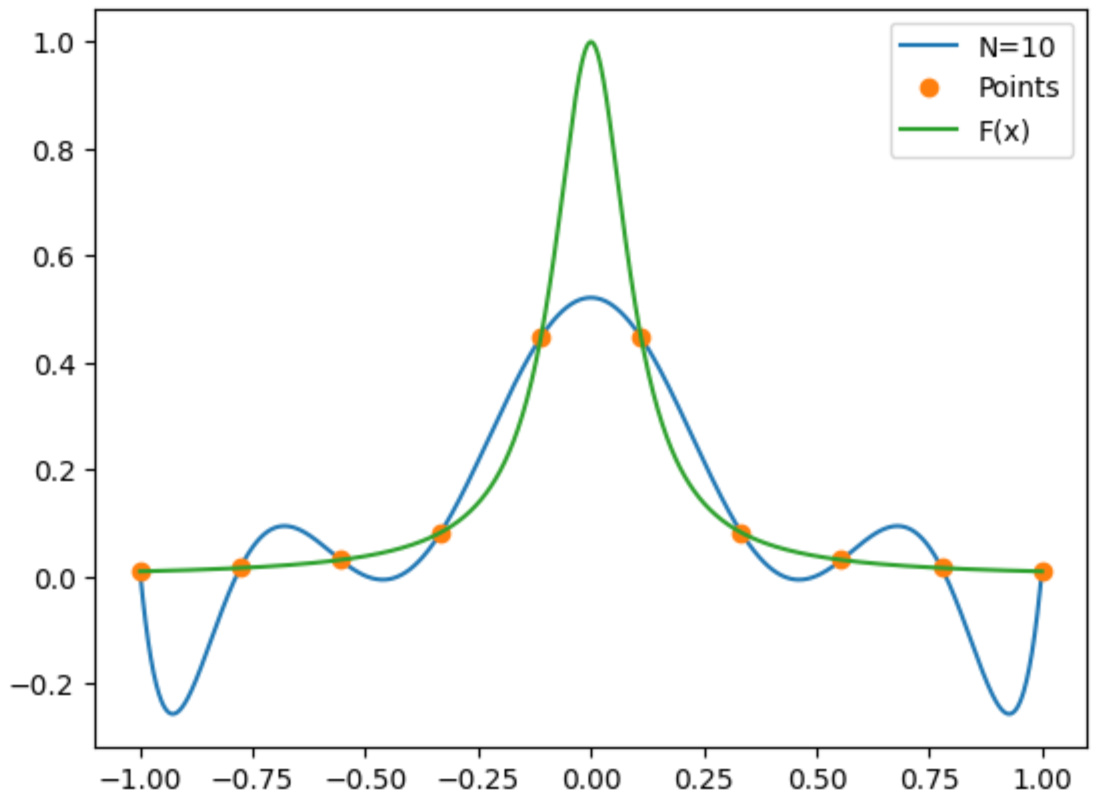
```
At N= 8 The max value is: 0.3819283437466645
```



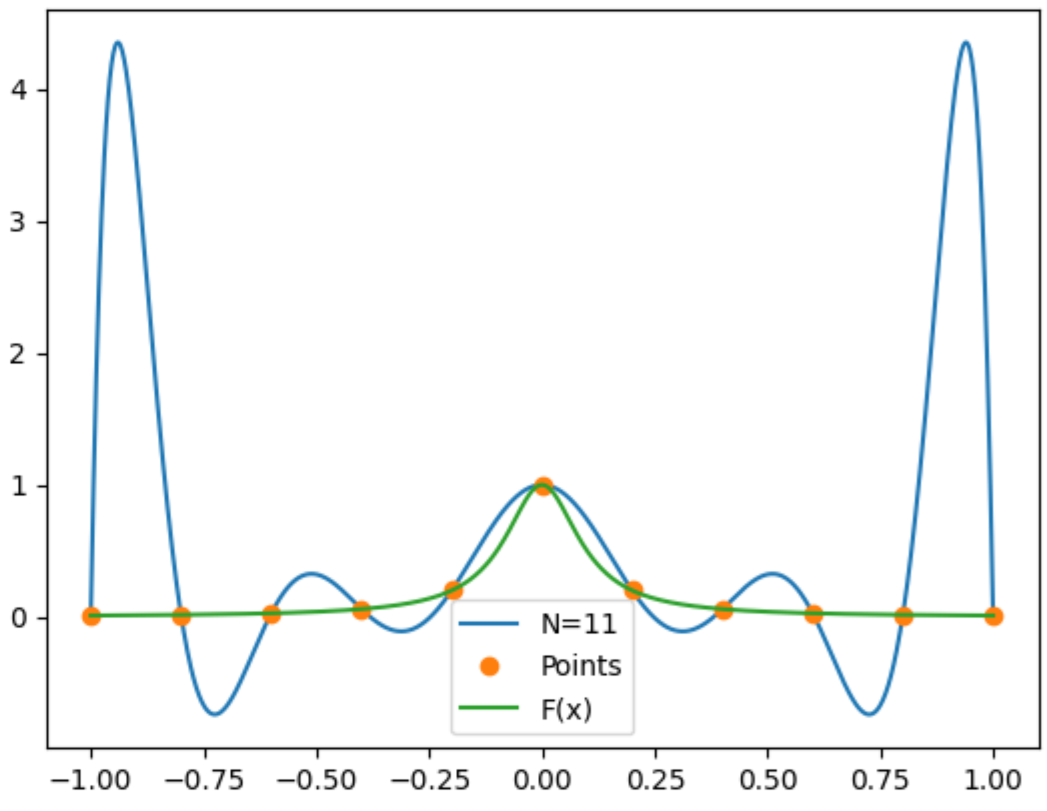
```
At N= 9 The max value is: 0.9999999999999959
```



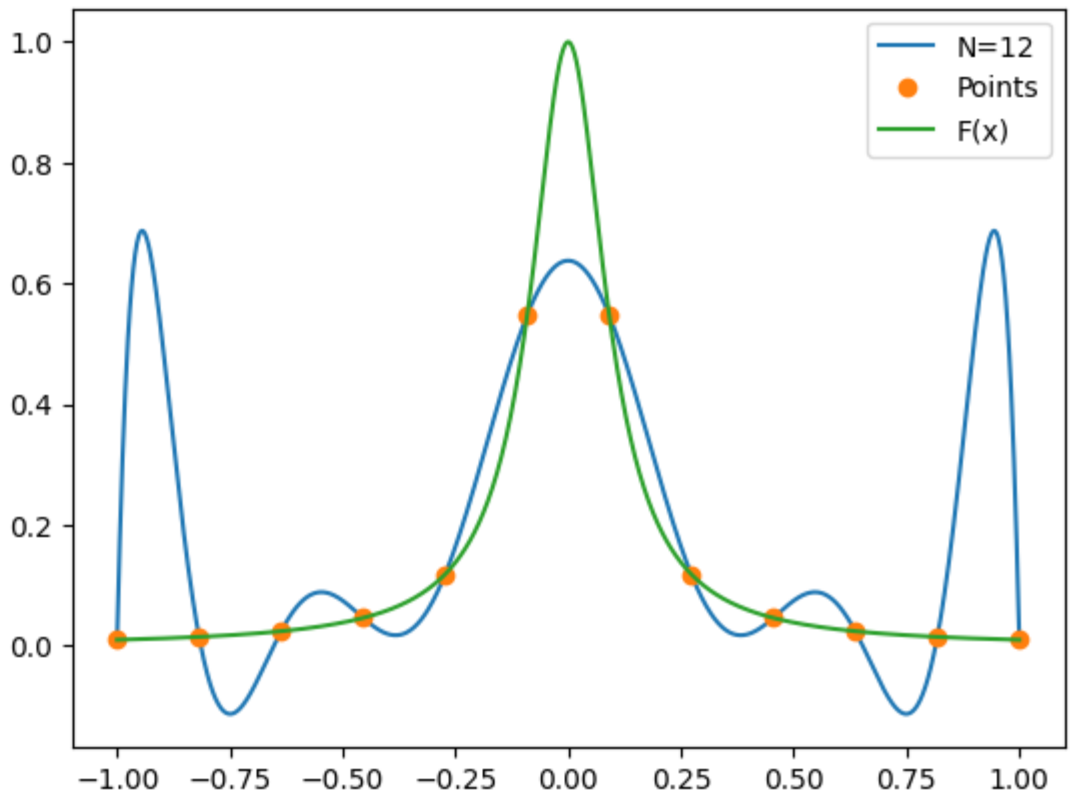
```
At N= 10 The max value is: 0.5218049284237937
```



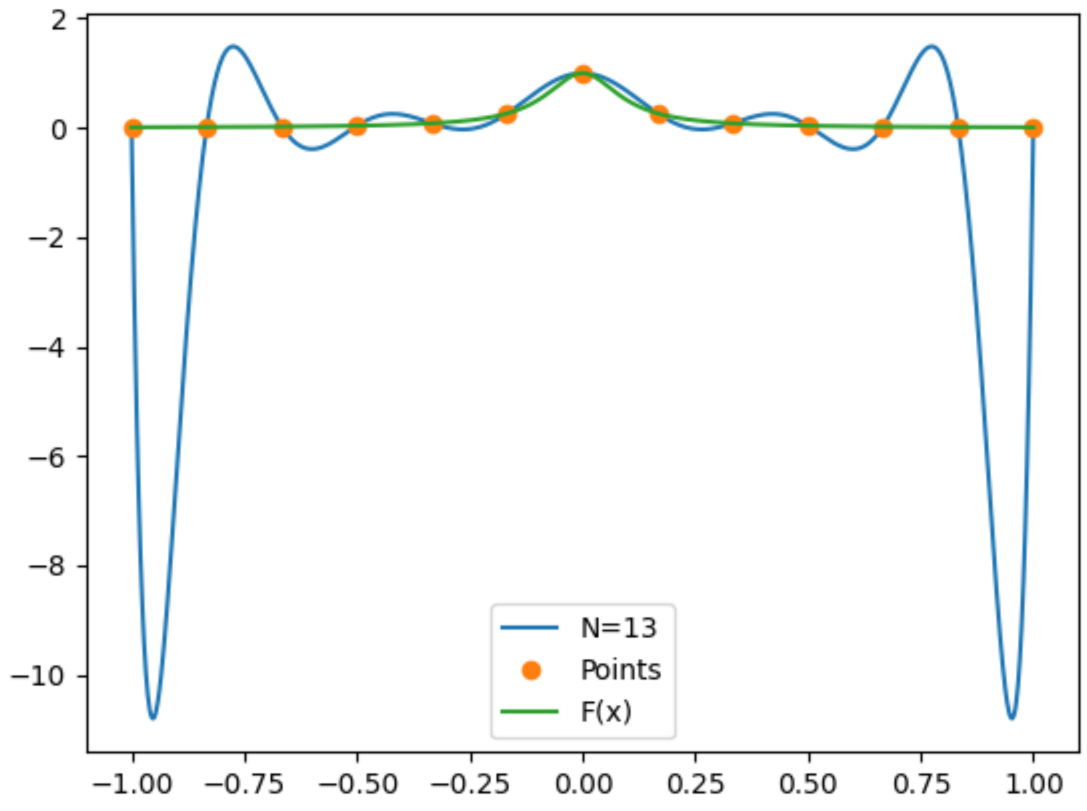
```
At N= 11 The max value is: 4.351515169127424
```



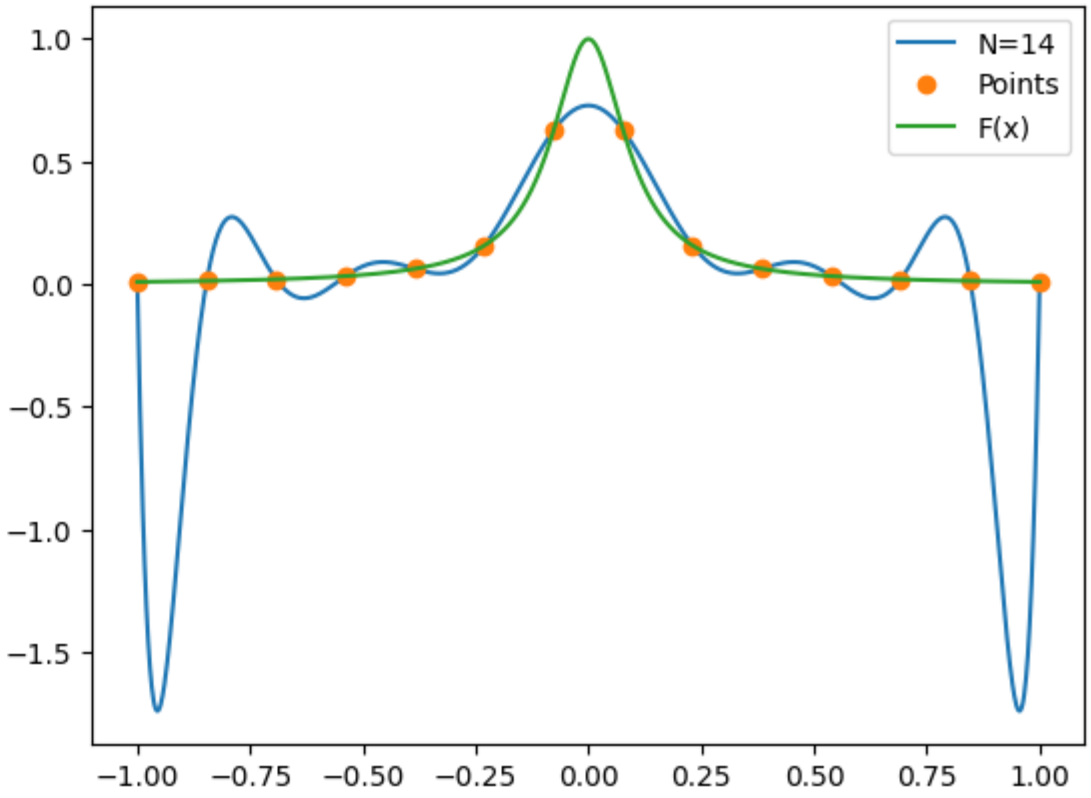
```
At N= 12 The max value is: 0.6872844686330045
```



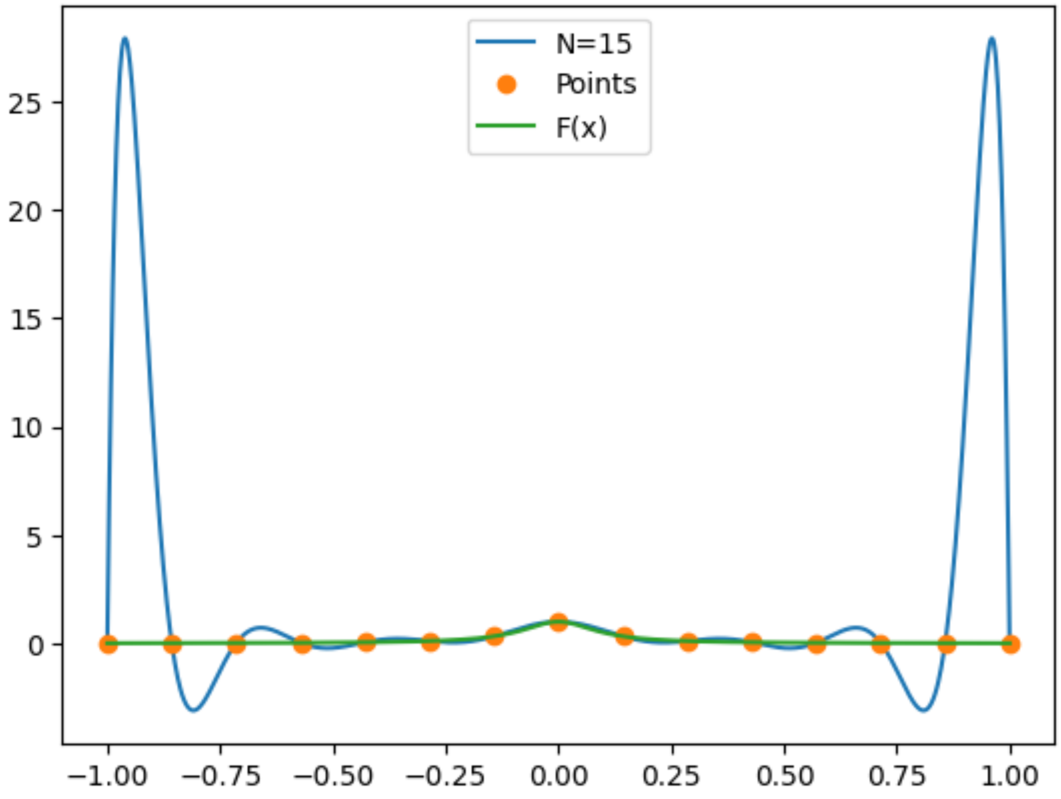
```
At N= 13 The max value is: 1.4877279683334592
```



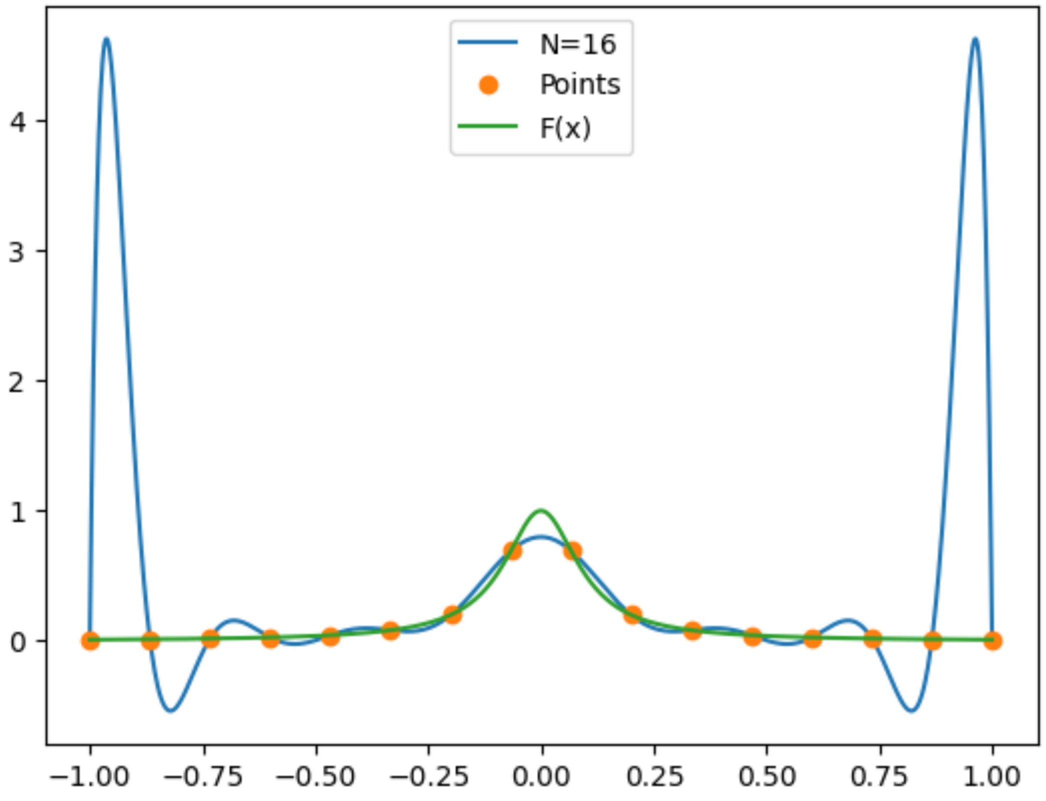
```
At N= 14 The max value is: 0.7289405174432307
```



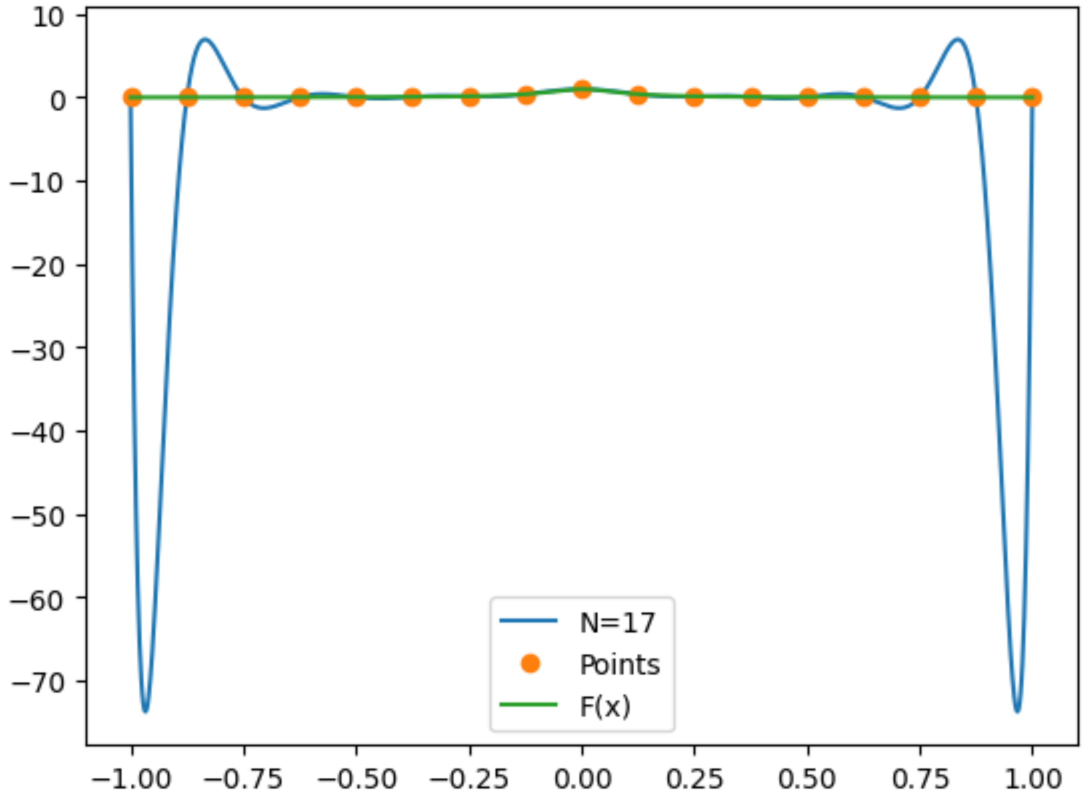
```
At N= 15 The max value is: 27.908207611999387
```



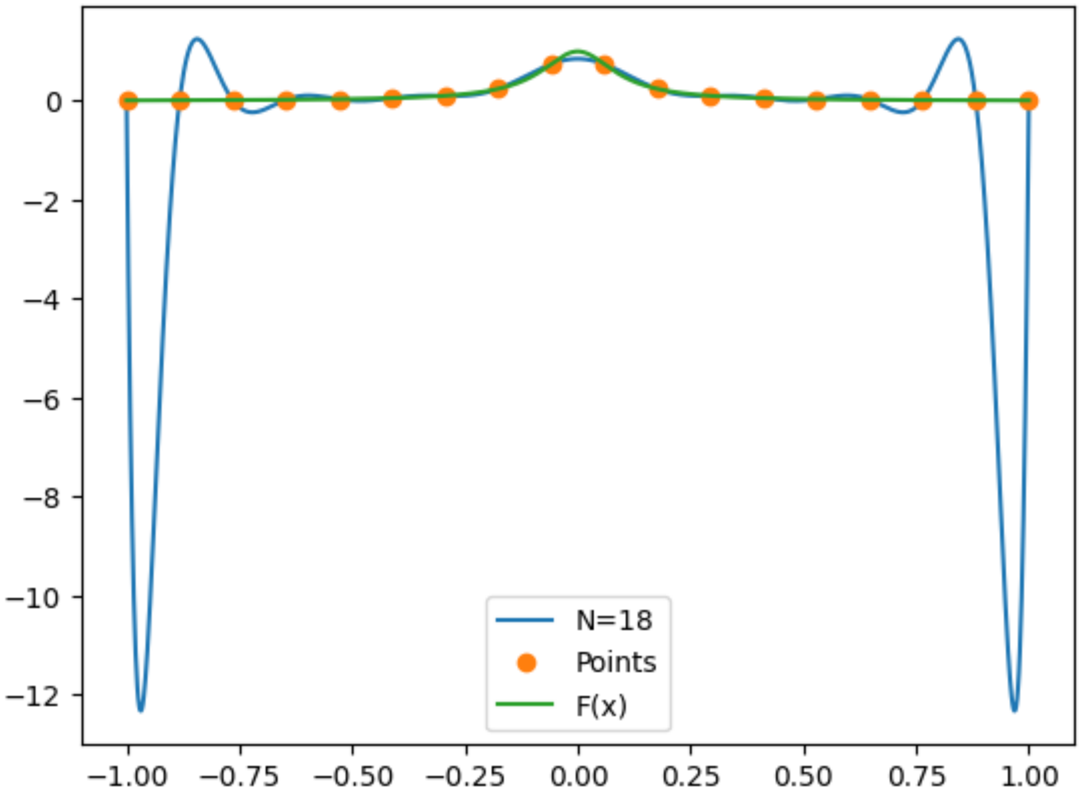
```
At N= 16 The max value is: 4.621874191009424
```



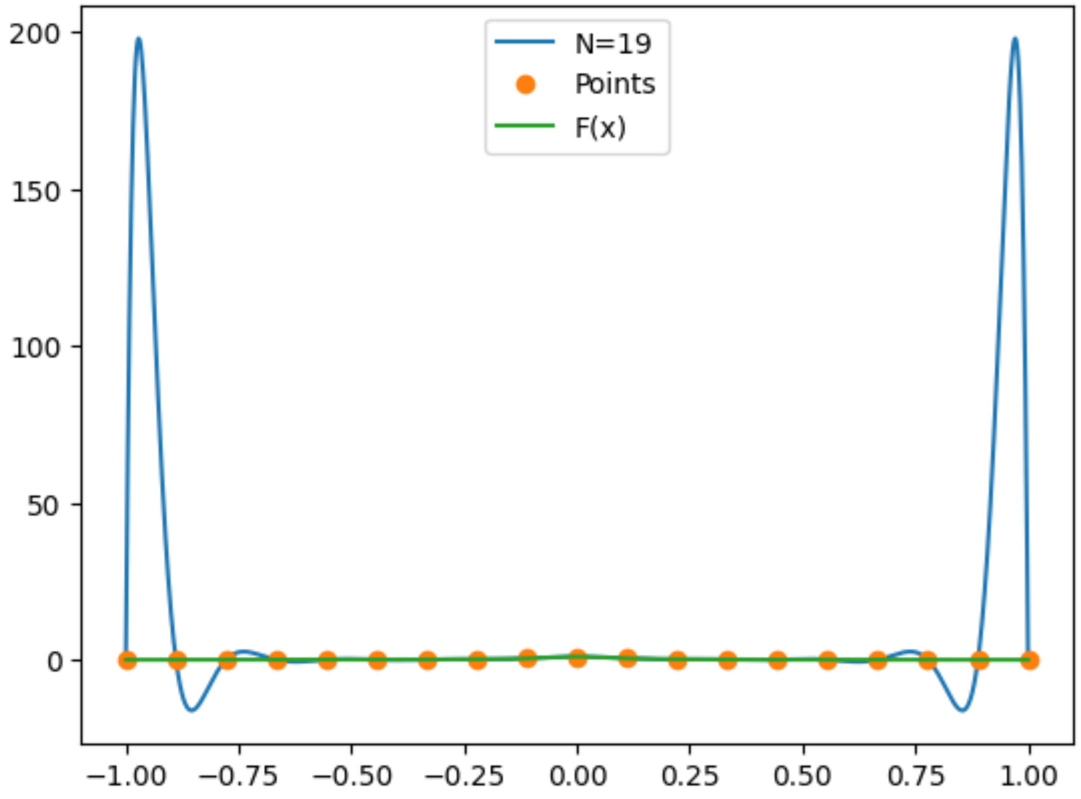
```
At N= 17 The max value is: 6.960564572897965
```



```
At N= 18 The max value is: 1.2514105362915586
```



```
At N= 19 The max value is: 198.09925590845523
```



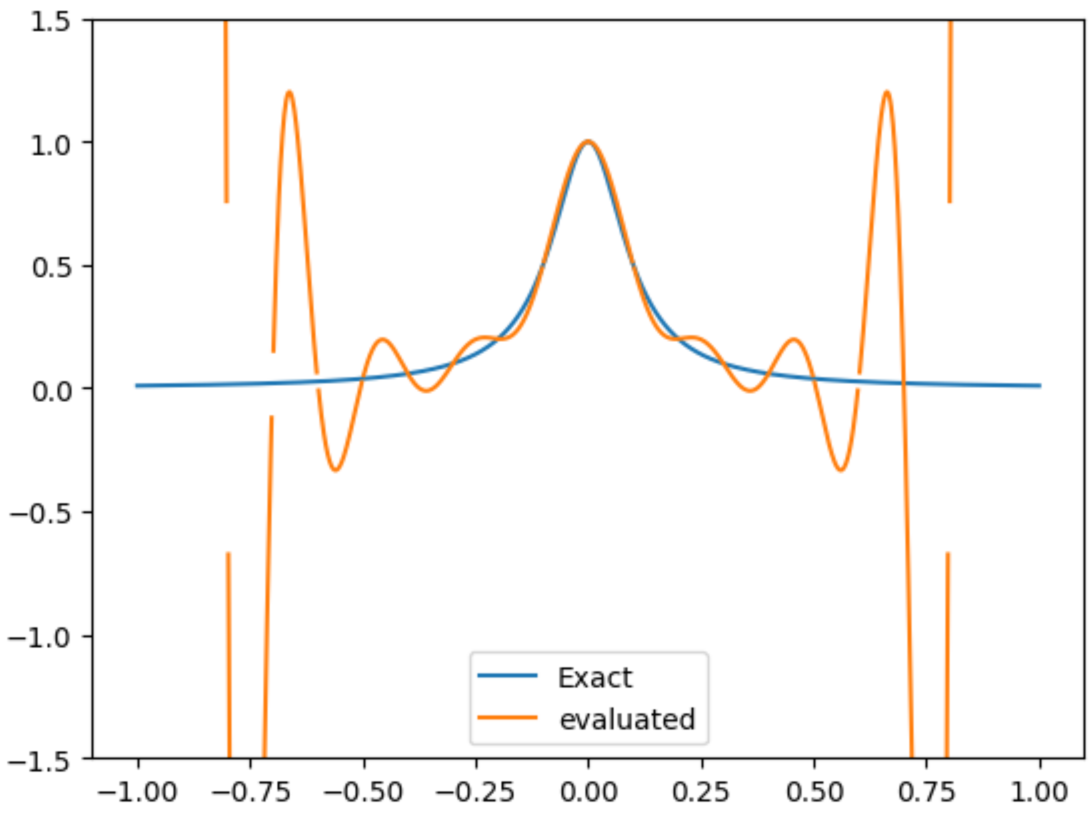
Here it is obvious that as N increases, the Runge Phenomenon increases in magnitude and creates issues with edge cases of the area. Eventually creating such large spikes that all interesting parts of the data have been lost as their magnitude is much less than the magnitude of variation at the edges.

Problem 2

In [9]:
```
"""Evaluate Lagrange"""
def lagrangeEval(f,N,a,b):
    xint = np.linspace(a,b,N+1)
    yint = f(xint)

    Neval = 1000
    xeval = np.linspace(a,b,Neval+1)
    yeval_l= np.zeros(Neval+1)

    for i in range(Neval+1):
        yeval_l[i]= lag.eval_lagrange(xeval[i],xint,yint,N)
    y = f(xeval)
    plt.plot(xeval,y, label = "Exact")
    plt.plot(xeval,yeval_l, label = "evaluated")
    plt.ylim(-1.5,1.5)
    plt.legend()
```

In [10]:
```
f = lambda x: 1/(1+(10*x)**2)
N=20
a=-1
b=1
lagrangeEval(f,N,a,b)
```

```
c:\Users\xman7\OneDrive - UCB-O365\Documents\APPM4600\Homework\Homework7\interp.py:2
4: RuntimeWarning: invalid value encountered in scalar divide
  peval = peval + phi*wj[j]*yint[j]/(xeval-xint[j])
```

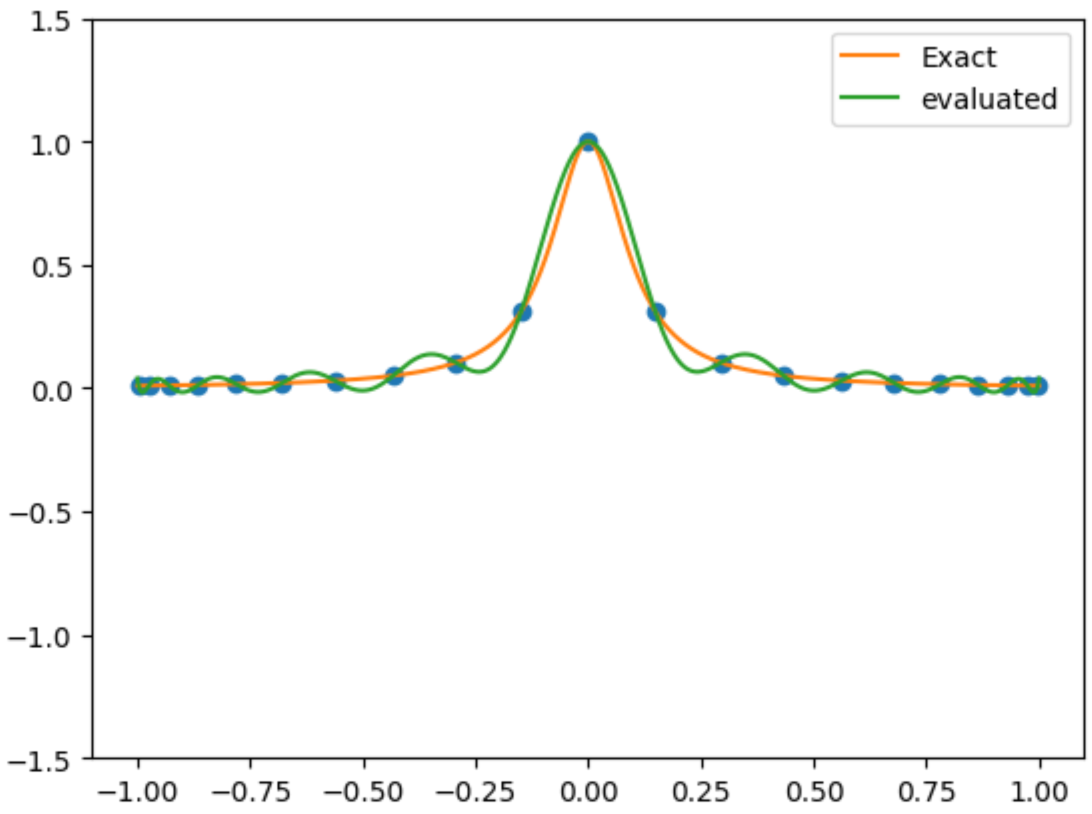


This still exhibits the bad behaviour, but shows good behaviour for $x << |bound|$

Problem 3

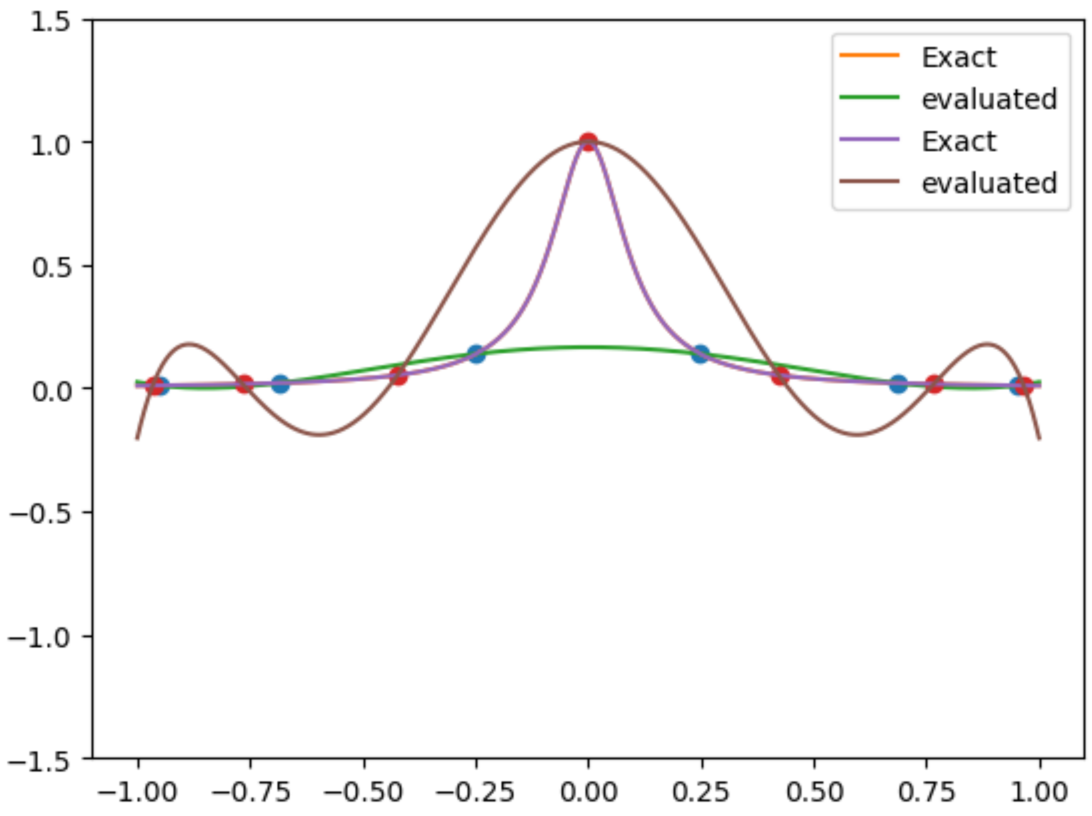
In [18]:
```
def lagrangeEval2(f,N,a,b):
    jint = np.linspace(1,N,N+1)
    xint = np.cos(np.pi*(2*jint-1)/(2*N))
    yint = f(xint)
    plt.plot(xint,yint,'o')

    Neval = 1000
    xeval = np.linspace(a,b,Neval+1)
    yeval_l= np.zeros(Neval+1)

    for i in range(Neval+1):
        yeval_l[i]= lag.eval_lagrange(xeval[i],xint,yint,N)
    y = f(xeval)
    plt.plot(xeval,y, label = "Exact")
    plt.plot(xeval,yeval_l, label = "evaluated")
    plt.legend()
```

In [ ]:
```
f = lambda x: 1/(1+(10*x)**2)
N=20
a=-1
b=1
lagrangeEval2(f,N,a,b)
```



Here you can see that using the chebyshev points the interpolation works wonderfully. So to get it to fail we will drastically decrease the number of points

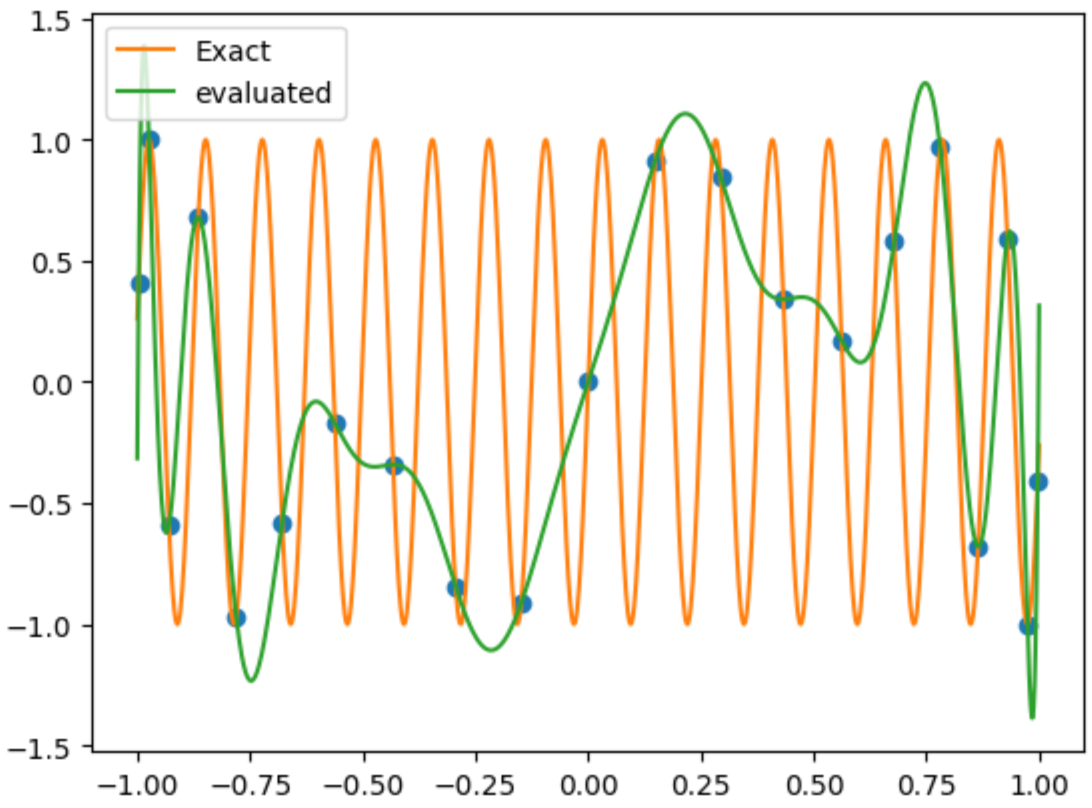
In [15]:
```
f = lambda x: 1/(1+(10*x)**2)
N3=5
a=-1
b=1
lagrangeEval2(f,N3,a,b)
lagrangeEval2(f,N3+1,a,b)
```



This leads to a lack of accuracy for small x if the number of points is even, but for odd numbers it works fine.

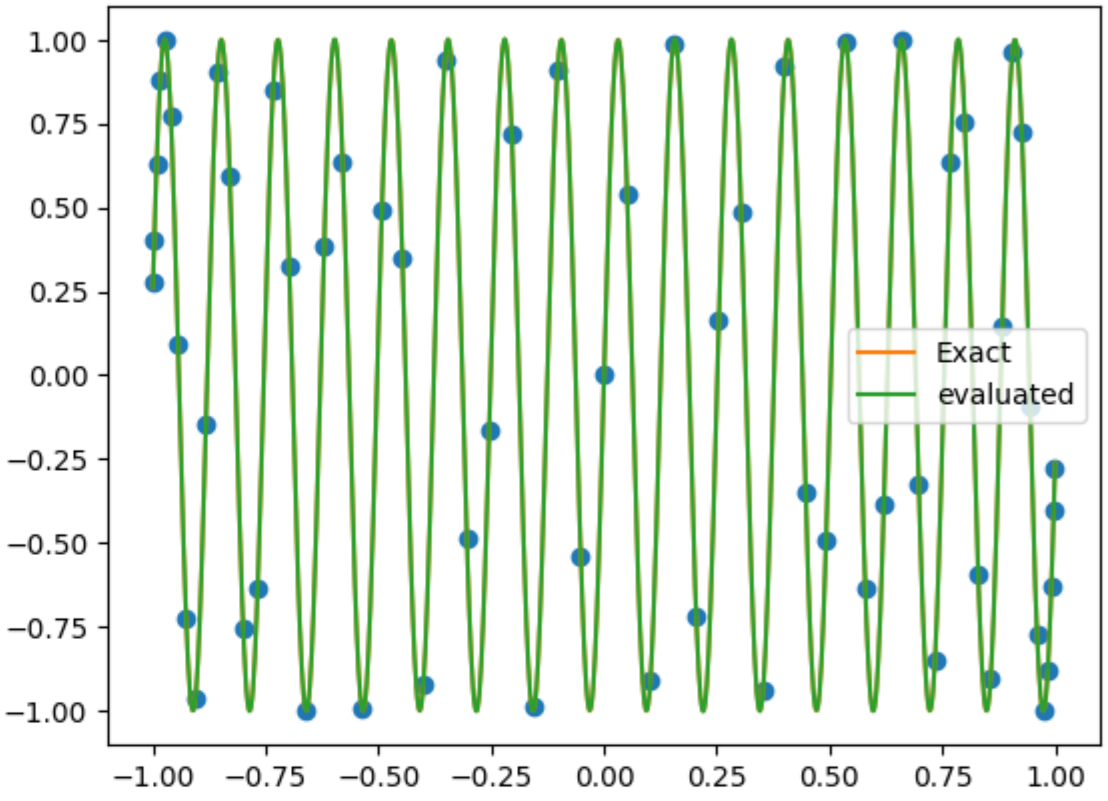
If we change f and it's bounds, maybe the interpolation won't work as well:

```
In [27]: f2 = lambda x: np.sin(50*x)
         lagrangeEval2(f2,N,a,b)
```



For a high frequency sinusoidal function this breaks for a reasonable N. You need $N \approx w$ where w is the frequency of your sinusoid.

In [29]:
```
f2 = lambda x: np.sin(50*x)
lagrangeEval2(f2,60,a,b)
```



In [ ]: