Homework 8

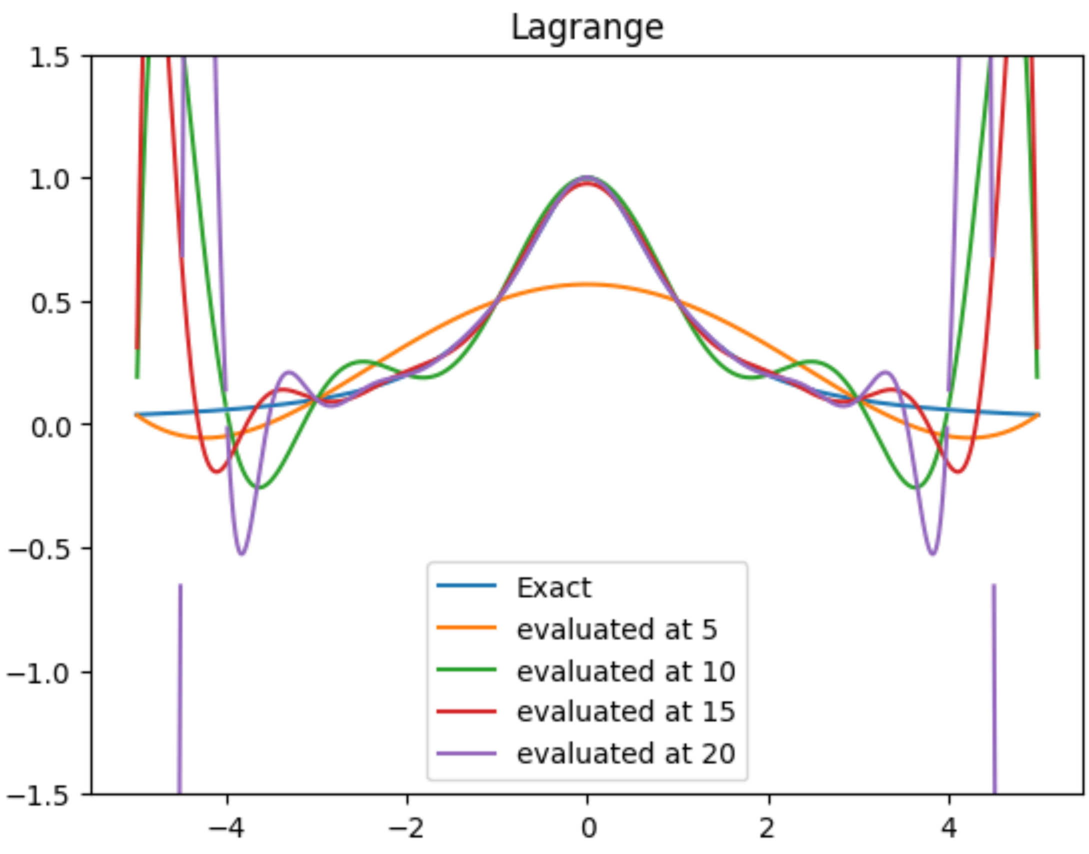
Problem 1

In [43]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as scipy
import interp as lag
import cubic_spline_demo as cube
import hermite_int as hermy

def lagrangeEval(f,N,a,b,cheby = 'False'):
    xint = np.linspace(a,b,N+1)
    if cheby == 'True':
        jint = np.linspace(1,N,N+1)
        xint = 5*np.cos(np.pi*(2*jint-1)/(2*N))

    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)
    if N==5:

        plt.plot(xeval,y, label = "Exact")
    plt.plot(xeval,yeval_l, label = "evaluated at "+str(N))
    plt.ylim(-1.5,1.5)
    plt.title("Lagrange")
    plt.legend()
```

Lagrange:

In [44]:
```
f = lambda x: 1/(1+x**2)
N=[5,10,15,20]
a=-5
b=5
for n in N:

    lagrangeEval(f,n,a,b)
```

```
c:\Users\xman7\OneDrive - UCB-O365\Documents\APPM4600\Homework\Homework8\interp.py:2
4: RuntimeWarning: invalid value encountered in scalar divide
  peval = peval + phi*wj[j]*yint[j]/(xeval-xint[j])
```

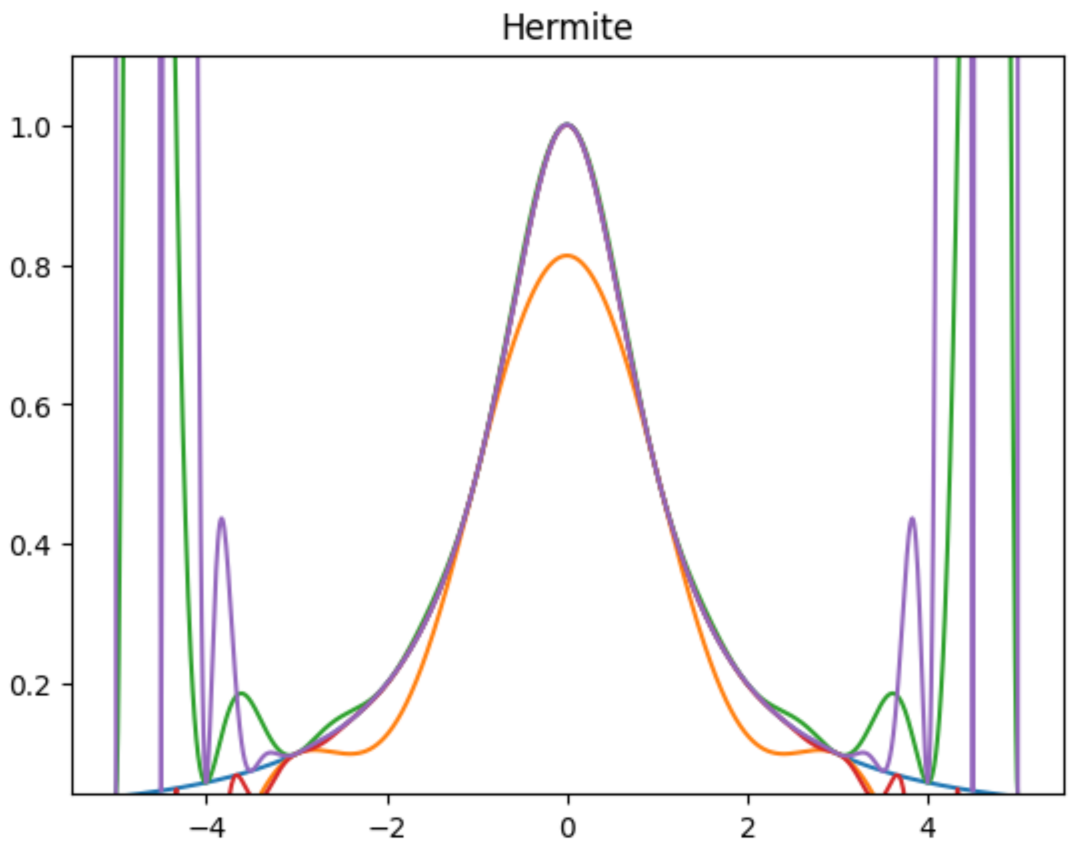


Hermite:

In [64]:
```
def hermite(f,fp,N,a,b,cheby = 'False'):

    xint = np.linspace(a,b,N+1)
    if cheby == 'True':
        jint = np.linspace(1,N,N+1)
        xint = 5*np.cos(np.pi*(2*jint-1)/(2*N))

    yint = np.zeros(N+1)
    ypint = np.zeros(N+1)
    for jj in range(N+1):
        yint[jj] = f(xint[jj])
        ypint[jj] = fp(xint[jj])

    Neval = 1000
    xeval = np.linspace(a,b,Neval+1)
    yevalH = np.zeros(Neval+1)
    for kk in range(Neval+1):
      yevalH[kk] = hermy.eval_hermite(xeval[kk],xint,yint,ypint,N)

    fex = np.zeros(Neval+1)
    for kk in range(Neval+1):
        fex[kk] = f(xeval[kk])


    if N==5:

        plt.plot(xeval,fex)
```

```
    plt.plot(xeval,yevalH,label='N='+str(N))
    plt.title("Hermite")
    plt.ylim(min(fex)*1.1,max(fex)*1.1)
```

In [65]:
```
fp = lambda x: -2*x/(1.+x**2)**2
for n in N:
    hermite(f,fp,n,a,b)
```

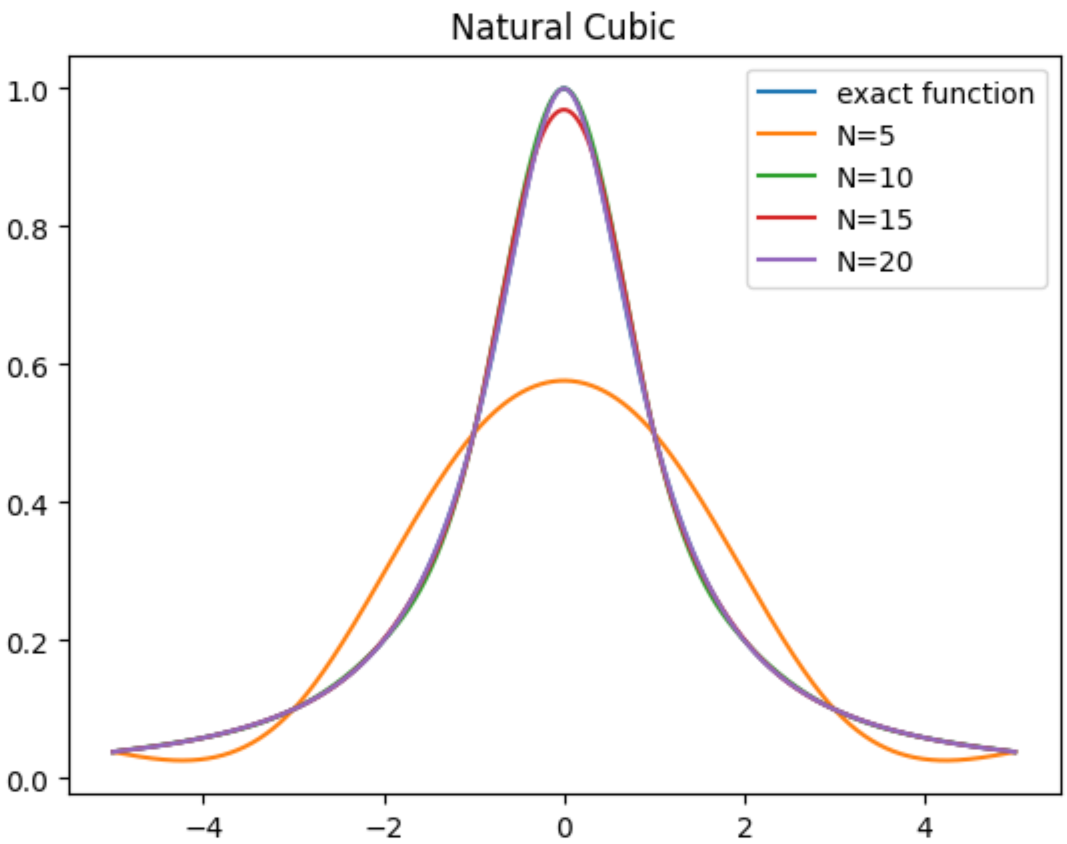


Natural Cubic:

In [71]:
```
def natural_spline(f,N,a,b, cheby = 'False'):
    xint = np.linspace(a,b,N+1)
    if cheby == 'True':
        jint = np.linspace(1,N,N+1)
        xint = 5*np.cos(np.pi*(2*jint-1)/(2*N))
        xint = np.sort(xint)
    yint = f(xint)

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

    e,h,g = cube.create_natural_spline(yint,xint,N)
    yeval = cube.eval_cubic_spline(xeval,Neval,xint,N,e,h,g)
    fex = f(xeval)

    if N == 5:
```

```
        plt.plot(xeval,fex,label='exact function')
    plt.plot(xeval,yeval,label='N='+str(N))
    plt.legend()
    plt.title("Natural Cubic")
```

In [72]:
```
for n in N:
    natural_spline(f,n,a,b)
```

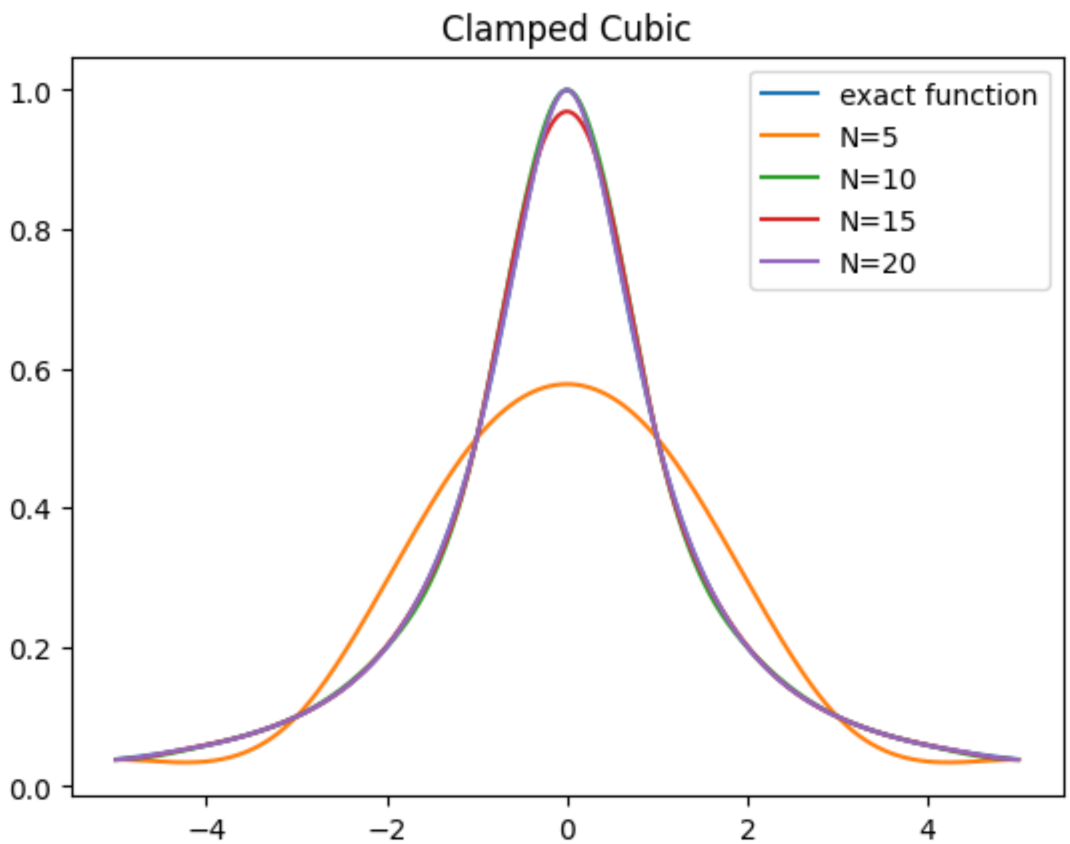


Clamped Cubic:

In [75]:
```
def cubicSpline(f,N,a,b, cheby = 'False'):
    xint = np.linspace(a,b,N+1)
    if cheby == 'True':
        jint = np.linspace(1,N,N+1)
        xint = 5*np.cos(np.pi*(2*jint-1)/(2*N))
        xint = np.sort(xint)
    yint = f(xint)

    Neval = 1000
    xeval = np.linspace(a,b,Neval+1)
    fex = f(xeval)


    cs = scipy.interpolate.CubicSpline(xint,yint,bc_type='clamped')
    if N==5:

        plt.plot(xeval,fex,label='exact function')
```

```
    plt.plot(xeval,cs(xeval), label = "N="+str(N))
    plt.legend()
    plt.title("Clamped Cubic")
```

In [56]:
```
for n in N:
    cubicSpline(f,n,a,b)
```

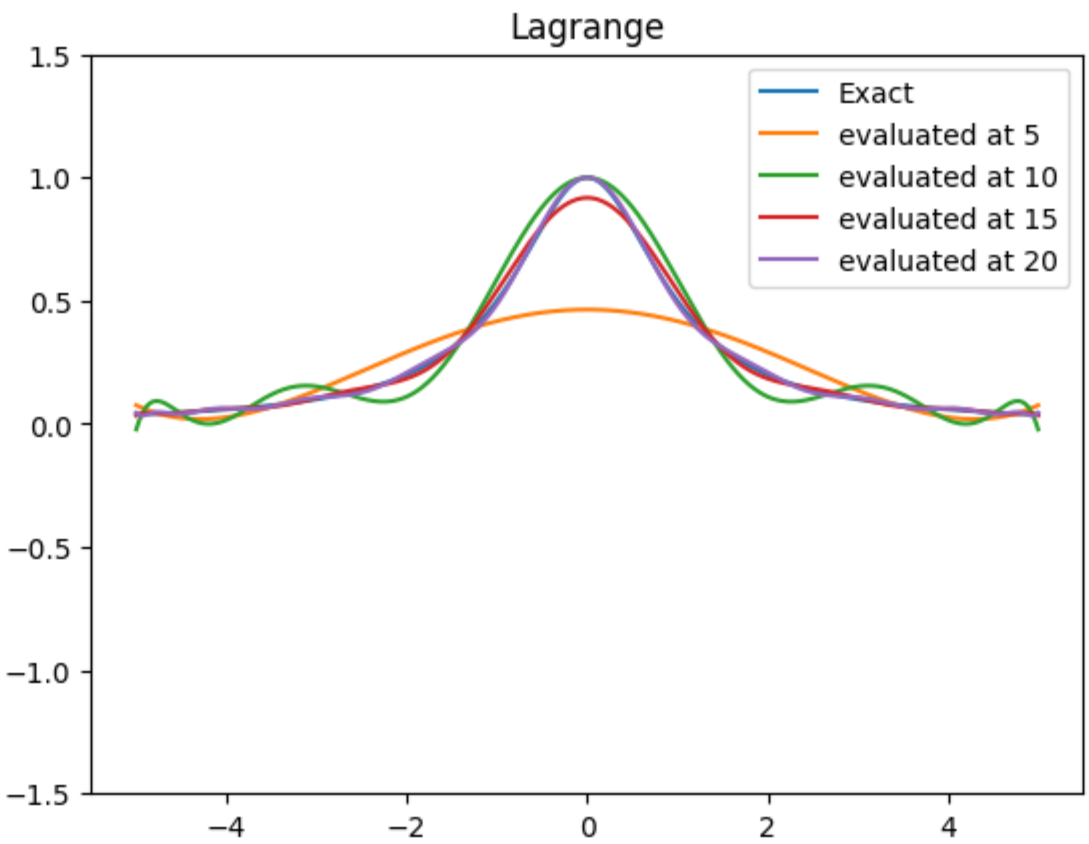


Clamped cubic spline seems to perform the best in this case. I would suspect this is due to the increased boundary conditions allowing the end points to be accurate to the original function.

Problem 2

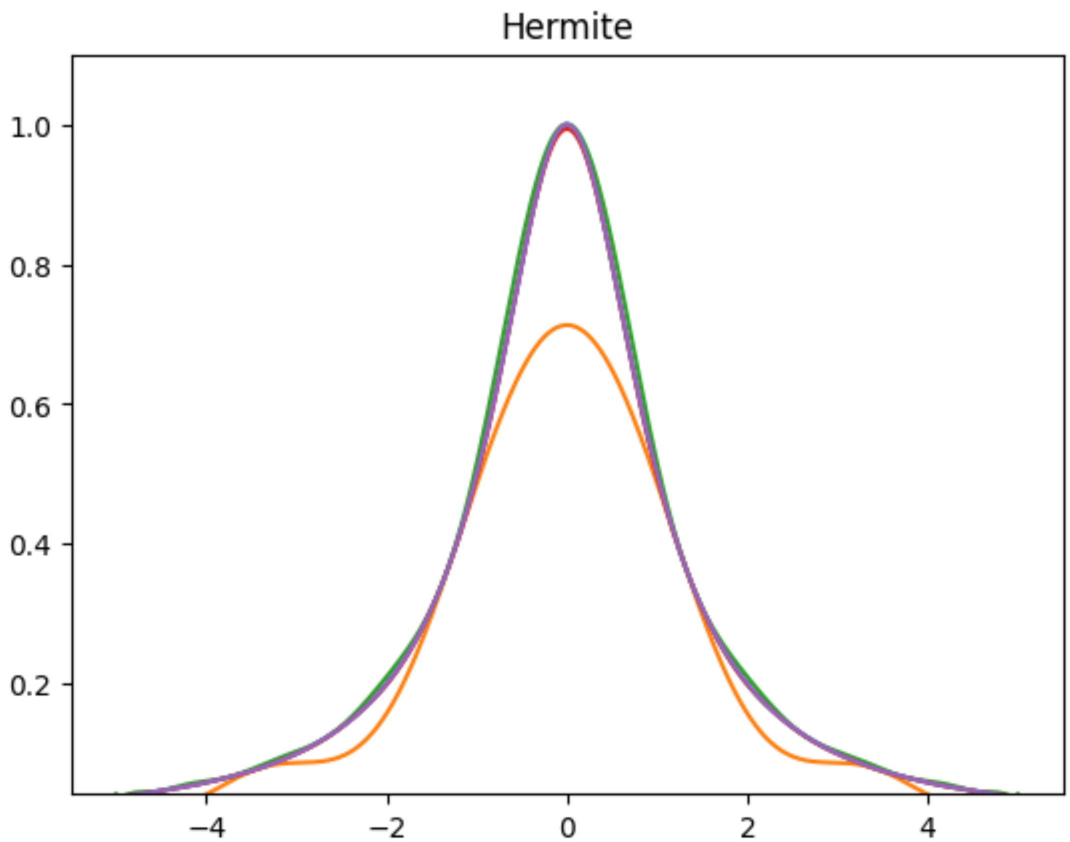
Lagrange:

In [57]:
```
for n in N:

    lagrangeEval(f,n,a,b, 'True')
```



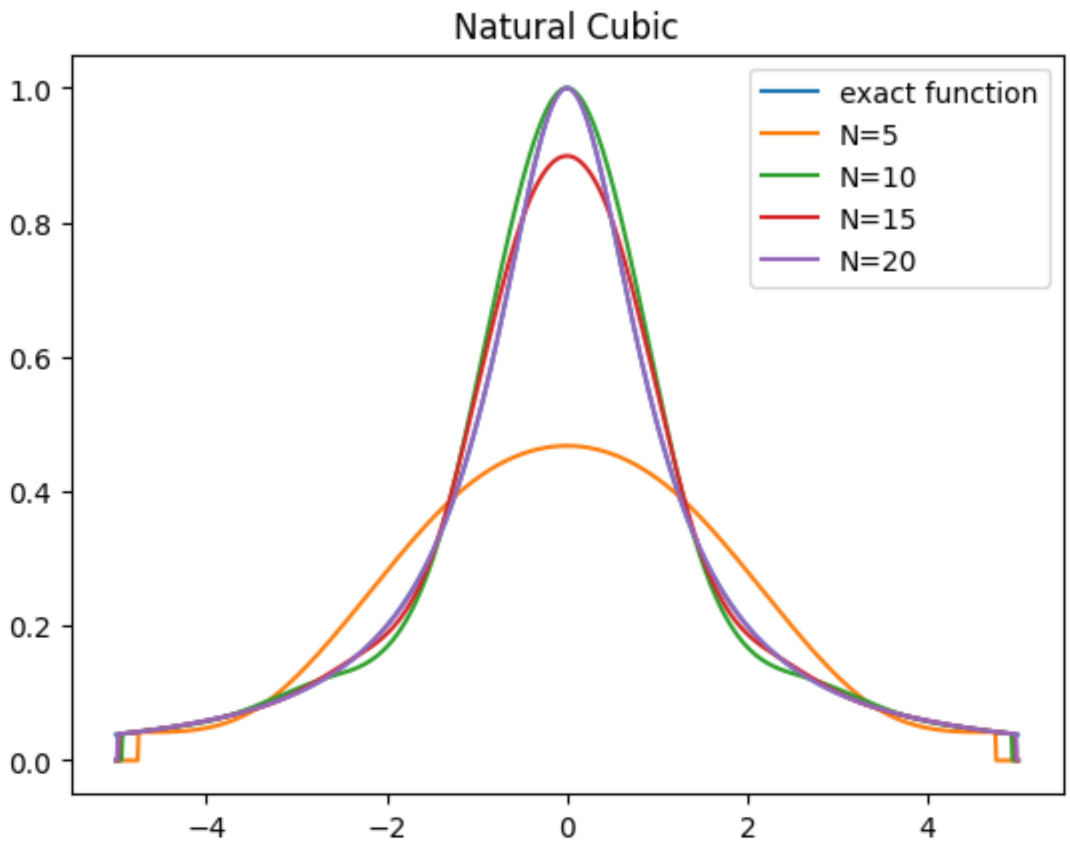
Hermite:

In [66]:
```
for n in N:

    hermite(f,fp,n,a,b, 'True')
```



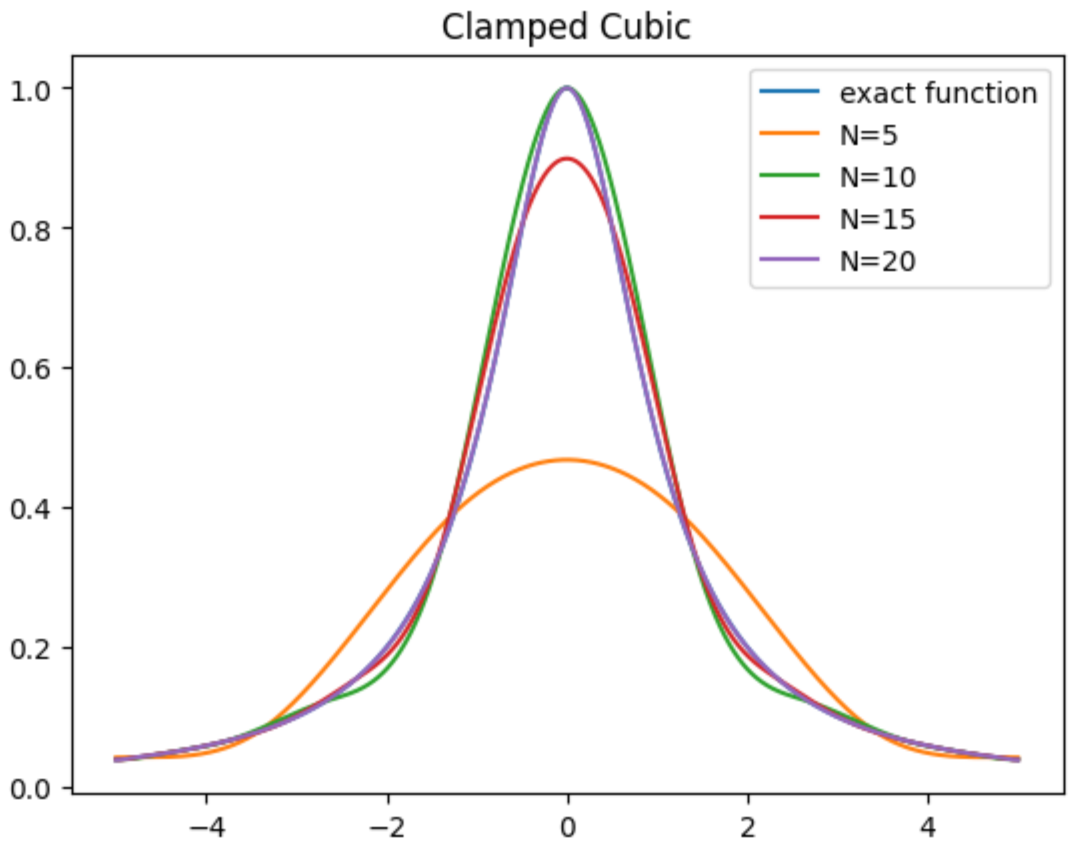
Natural Cubic:

In [73]:
```
for n in N:

    natural_spline(f,n,a,b, 'True')
```



Clamped Cubic:

In [76]:
```
for n in N:

    cubicSpline(f,n,a,b, 'True')
```



This change of using chebychev nodes instead of equidistanced ones creates more issues in lower N values, but increases the accuracy very quickly, such that at N=10 a good approximation is made overall for all conditions, and that approximation is much better than the N=10 for equidistanced nodes.

Problem 3:

To approximate a periodic function, the function and it's first two derivatives must repeat over the period. In other words $F(a) = F(b)$ & $F'(a) = F'(b)$ & $F''(a) = F''(b)$

In [ ]: