Homework 10

Due date: Friday, April 15 at 11:59pm in Gradescope

Use the following format for homework filename:

`Homework_10.py`

This assignment is worth 26 points.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

try:
    %matplotlib inline
except:
    pass

def trapezoid_method(a,b,n,f):
    N = int(n)
    xvals = np.linspace(a,b,N+1)
    fvals = f(xvals)
    dx = (b-a)/N
    return dx/2.*(fvals[0] + fvals[N] + 2.*np.sum(fvals[1:N]))

def simpson_method(a,b,n,f):
    N = int(n)
    xvals = np.linspace(a,b,N+1)
    fvals = f(xvals)
    dx = (b-a)/N
    return dx/3.*(fvals[0] + fvals[N] + 2.*np.sum(fvals[2:N-1:2]) + 4.*np.su

```

Problem 1 (3 points)

Given that

$$F = \frac{A}{\int_0^{T_1} c(t)\,dt + \int_{T_1}^{T_2} m(t)\,dt},$$

where $A = 3, T_1 = \pi/2, T_2 = \pi/4,$

$$c(t) = \cos(t) - sin(t) \quad \text{and} \quad m(t) = \sin(t) - cos(t).$$

Use Trapezoid Method to compute the denominator of the above formula and then compute F. Print your result for F.

In [63]:
```
# Problem 1 solution
def c(t):
    return np.cos(t) - np.sin(t)
def m(t):
    return np.sin(t) - np.cos(t)

def trapMethod(y, a, b, n):
    h = (b - a) / n
    bigSum = (y(a) + y(b))

    i = 1
    while i < n:
        bigSum += 2 * y(a + i * h)
        i += 1

    return ((h / 2) * bigSum)

A = 3

D = trapMethod(c, 0, np.pi / 2, 100) + trapMethod(m, np.pi / 4, np.pi / 2, 1
print("Denominator", D)

F = A / D
print("F", F)
```

```
Denominator 0.4142114331396567
F 7.242677917556446
```

Problem 2 (2 points)

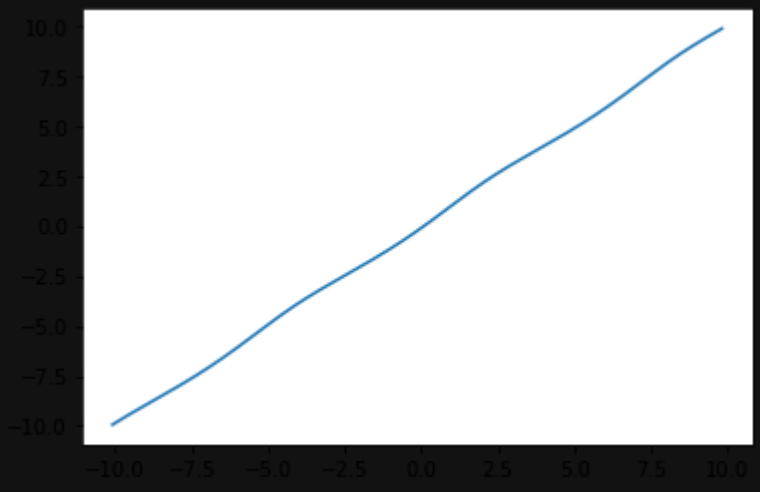
Write a program to graph several members of the family of curves with parametric equations

$$\begin{cases} x = t + a\cos(t) \\ y = t + a\sin(t) \end{cases}$$

where $a > 0$. How does the shape change as a increases? For what values of a does the curve have a loop?

In [54]:
```
# Problem 2 solution
def familyCurves(a, t):
    x = []
    y = []

    for i in a:
        arr1 = []
        arr2 = []

        for ii in t:
            arr1.append(ii + i * np.cos(ii))
            arr2.append(ii + i * np.sin(ii))
            x.append(arr1)
            y.append(arr2)

    return x, y

a = np.arange(0.1,10,0.2)
t = np.arange(-10,10,0.1)

x, y = familyCurves(a, t)

print (plt.plot(x[8], y[8]))
print ("Loop at", (a[8]))
```

```
[<matplotlib.lines.Line2D object at 0x0000024ACA771850>]
Loop at 1.7000000000000004
```



As a increases, the size of the loop becomes larger as well. The curve forms a loop when $a > 1$.

Problem 3

a) (3 points)

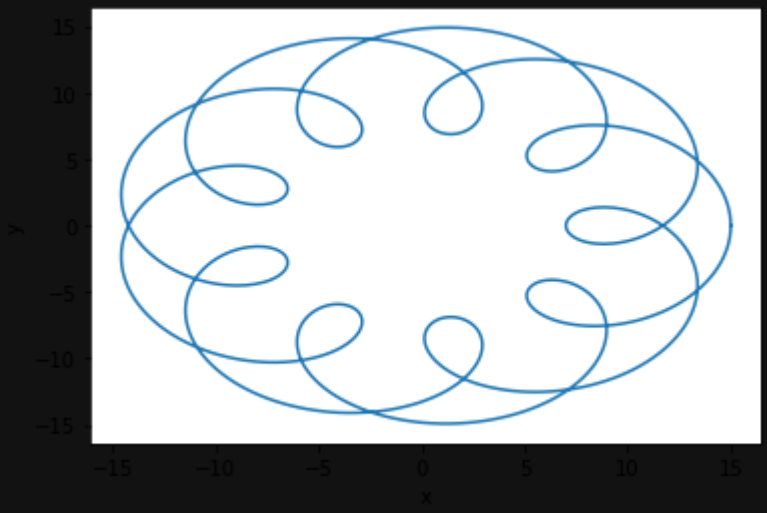
Write a program to graph the *epitrochoid* curve with parametric equations

$$\begin{cases} x(t) = 11\cos(t) - 4\cos\left(\frac{11t}{2}\right) \\ y(t) = 11\sin(t) - 4\sin\left(\frac{11t}{2}\right) \end{cases}$$

Generate the plot over the interval $-2\pi \leq t \leq 2\pi$.

In [56]:
```
# Problem 3a solution
t = np.linspace(-2 * np.pi, 2 * np.pi, 1000)
x = 11 * np.cos(t) - 4 * np.cos(11 * t / 2)
y = 11 * np.sin(t) - 4 * np.sin(11 * t / 2)

plt.plot(x , y)
plt.xlabel('x')
plt.ylabel('y')
```

Out[56]: Text(0, 0.5, 'y')



b) (3 points)

If a curve C is described by the parametric equations

$$x = f(t),\ y = g(t),\ \alpha \leq t \leq \beta,$$

where f' and g' are continuous on $[\alpha, \beta]$ and C is traversed exactly once as t increases from α to β, then the length of C is

$$L = \int_{\alpha}^{\beta} \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}\, dt$$

Use the Simpson's Method with a number of interval $N = 10$ to estimate the length of the curve in part **a)**.

In [32]:
```
# Problem 3b solution
dy = lambda t : 11 * np.cos(t) - 22 * np.cos(11 * t / 2)
dx = lambda t : -11 * np.sin(t) + 22 * np.sin(11 * t / 2)
fx = lambda t : np.sqrt(dy(t) ** 2 + dx(t) ** 2)

N = 10
t = np.linspace(-2 * np.pi, 2 * np.pi, N + 1)
h = t[1] - t[0]
i = 0
simpsAns = 0

while(i < N):
    simpsAns += (fx(t[i]) + 4 * fx(t[i + 1]) + fx(t[i + 2])) * (h / 3)
    i += 2

print("Integral", simpsAns)
```

```
Integral 293.8812771564153
```

Problem 4 (5pts)

An automobile of mass $M = 4500$ kg is moving at a speed of 35 m/s. The engine is disengaged suddenly at $t = 0$ sec. Assume that the equation of motion after $t = 0$ is given by

$$4500v\frac{dv}{dx} = -8.276v^2 - 2000$$

where $v = v(t)$ is the speed (m/sec) of the car at t. The left side represents $Mv\,(dv/dx)$. The first term on the right is the aerodynamic drag, and the second term is the rolling resistance of the tires. Calculate how far the car travels until the speed reduces to 15 m/sec. (Hint: The equation of motion may be integrated as:

$$\int_{15}^{35} \frac{4500v}{8.276v^2 + 2000}\,dv = \int dx = x$$

Evaluate the equation above using the Trapezoid Method.

In [31]:
```python
# Problem 4 solution
def func(x):
    return 4500 * x / (8.276 * x ** 2 + 2000)

def trapMethod(a, b, n):
    h = (b - a) / n
    bigSum = (func(b) + func(a)) / 2

    for k in range(1, n):
        bigSum = bigSum + func(a + h * k)

    return bigSum * h

a = 15
b = 35
n = 10000

print('Distance', trapMethod(a, b, n))
```

```
Distance 311.32925621091255
```

Problem 5 (2pts)

This problem is autograded.

Since the area of the unit circle is $A = \pi$, it follows that

$$\frac{\pi}{2} = \int_{-1}^{1} \sqrt{1 - x^2}\, dx.$$

Therefore we can approximate π by approximating this integral.

Write a function `pi_approx_trapz(dx)` with output `pi_approx` that uses the **Trapezoid Method** to compute approximate values of π in this way. Print your approximation results for $dx = 1,\ 1/2,\ 1/4$ and $1/8$ and comment on them.

In [30]:
```
# Problem 5 solution
def function(x):
    return np.sqrt(1 - x * x)

def TrapMethod(f, a, b, n):
    h = (b - a) / n
    s = 0.5 * (f(a) + f(b))

    for i in range(1, n, 1):
        s = s + f(a + i * h)

    return h * s

a = -1
b = 1
n = 2

ans = TrapMethod(function, a, b, n)
print(ans * 2)

n = 4
ans = TrapMethod(function, a, b, n)
print(ans * 2)

n = 8
ans = TrapMethod(function, a, b, n)
print(ans * 2)

n = 16
ans = TrapMethod(function, a, b, n)
print(ans * 2)
```

```
2.0
2.732050807568877
2.9957090681024403
3.0898191443571745
```

Problem 6 (2pts)

This problem is autograded.

Repeat Problem 5 but write a function `pi_approx_simps(dx)` with output `pi_approx` that uses **Simpson's Method** instead of the Trapezoid Method. See Lecture 10 and the recommended Calculus text in the Syllabus for material on Simpson's Method. Are the approximations for a given dx better than for the Trapezoid Method?

In [114]:
```
# Solution for Problem 6

```

Problem 7

Periodic functions can be written as an infinite sum of sine and cosine waves using the formula for Fourier series:

$$f(x) = \frac{A_0}{2} + \sum_{n=1}^{\infty} [A_n \cos(nx) + B_n \sin(nx)]$$

It can be shown that the values of A_n and B_n can be computed using the following formulas with $n \geq 0$:

$$A_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx)\, dx$$
$$B_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx)\, dx$$

Much like the Talyor series you're already familiar with, functions can be approximated by truncating the Fourier series at some $n = N$. Fourier series can be used to approximate some particularly nasty functions, such as the step function, and they form the basis of many engineering applications, such as signal processing.

a) (4 points)

This problem is autograded.

Write a function `my_fourier_coef(f,n)` with output `[An,Bn]`, where `f` is an function object that of a 2π-periodic function f. The function `my_fourier_coef` should compute the n-th Fourier coefficients, A_n and B_n, in the Fourier series for f defined by the two formulas given earlier. Use Simpson's method to perform the integration with at least 1000 points. Ensure that your function returns the results `[An,Bn]` as a list.

In [42]:
```
# Problem 7a solution
from scipy.integrate import quad
def func1(x):
    return np.cos(n * x) * f(x)

def func2(x):
    return np.sin(n * x) * f(x)

def my_fourier_coef(f, n):
    An = quad(func1, -np.pi, np.pi)[0]
    Bn = quad(func2, -np.pi, np.pi)[0]

    return [An / np.pi, Bn / np.pi]
```

b) (2 points)

Test your `my_fourier_coef` using the `plot_result` function given below with

1. $f(x) = \sin(\exp(x))$
2. $f(x) = \frac{x}{x^2+1}$
3. $f(x) = -x^3 \exp(-x^2)$

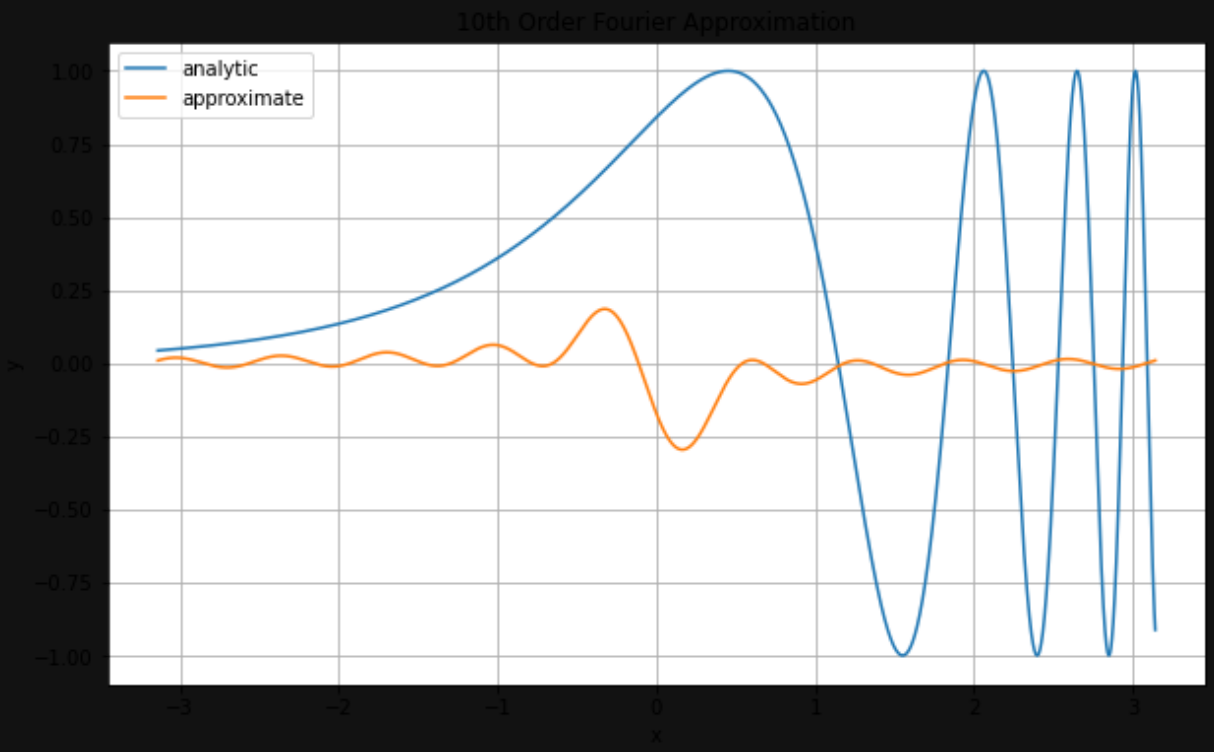
for $N = 10, 30$. Comment on the differences between the plots for the different N. One of the f's will be identical for both values of N, which one is it? Look at $f(-\pi)$ and $f(\pi)$ and conjecture as to why the Fourier approximation is better for that f than for the others.

In [43]:

```
def plot_results(f, N):
    x = np.linspace(-np.pi, np.pi, 10000)
    [A0, B0] = my_fourier_coef(f, 0)
    y = A0*np.ones(len(x))/2
    for n in range(1, N):
        [An, Bn] = my_fourier_coef(f, n)
        y += An*np.cos(n*x)+Bn*np.sin(n*x)
    plt.figure(figsize = (10,6))
    plt.plot(x, f(x), label = "analytic")
    plt.plot(x, y, label = "approximate")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.grid()
    plt.legend()
    plt.title(f"{N}th Order Fourier Approximation")
    plt.show()
```

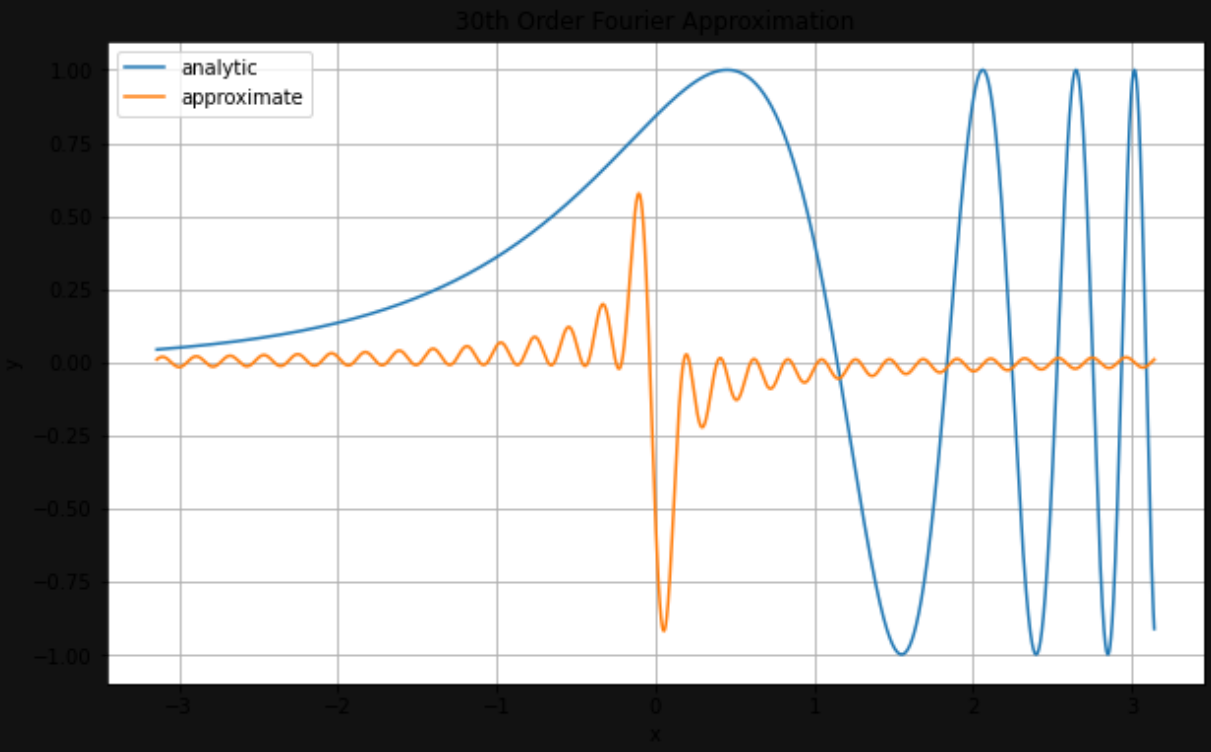
In [45]:
```
# Problem 7b solution
f = lambda x : np.sin(np.exp(x))
N = 10
plot_results(f, N)
```

```
C:\Users\austi\AppData\Local\Temp/ipykernel_15132/1227324076.py:9: IntegrationW
arning: The integral is probably divergent, or slowly convergent.
  An = quad(func1, -np.pi, np.pi)[0]
C:\Users\austi\AppData\Local\Temp/ipykernel_15132/1227324076.py:10: Integration
Warning: The integral is probably divergent, or slowly convergent.
  Bn = quad(func2, -np.pi, np.pi)[0]
```



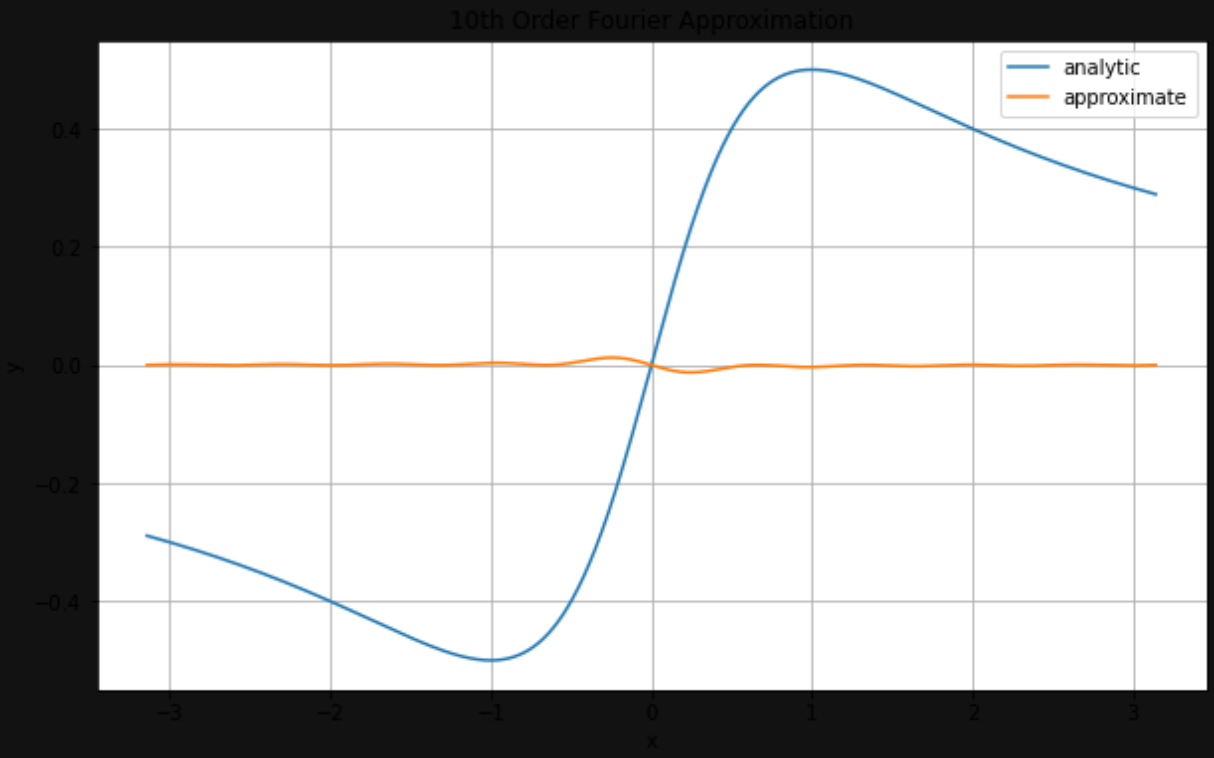
In [46]:
```
f = lambda x : np.sin(np.exp(x))
N = 30
plot_results(f, N)
```

```
C:\Users\austi\AppData\Local\Temp/ipykernel_15132/1227324076.py:9: IntegrationW
arning: The integral is probably divergent, or slowly convergent.
  An = quad(func1, -np.pi, np.pi)[0]
C:\Users\austi\AppData\Local\Temp/ipykernel_15132/1227324076.py:10: Integration
Warning: The integral is probably divergent, or slowly convergent.
  Bn = quad(func2, -np.pi, np.pi)[0]
```



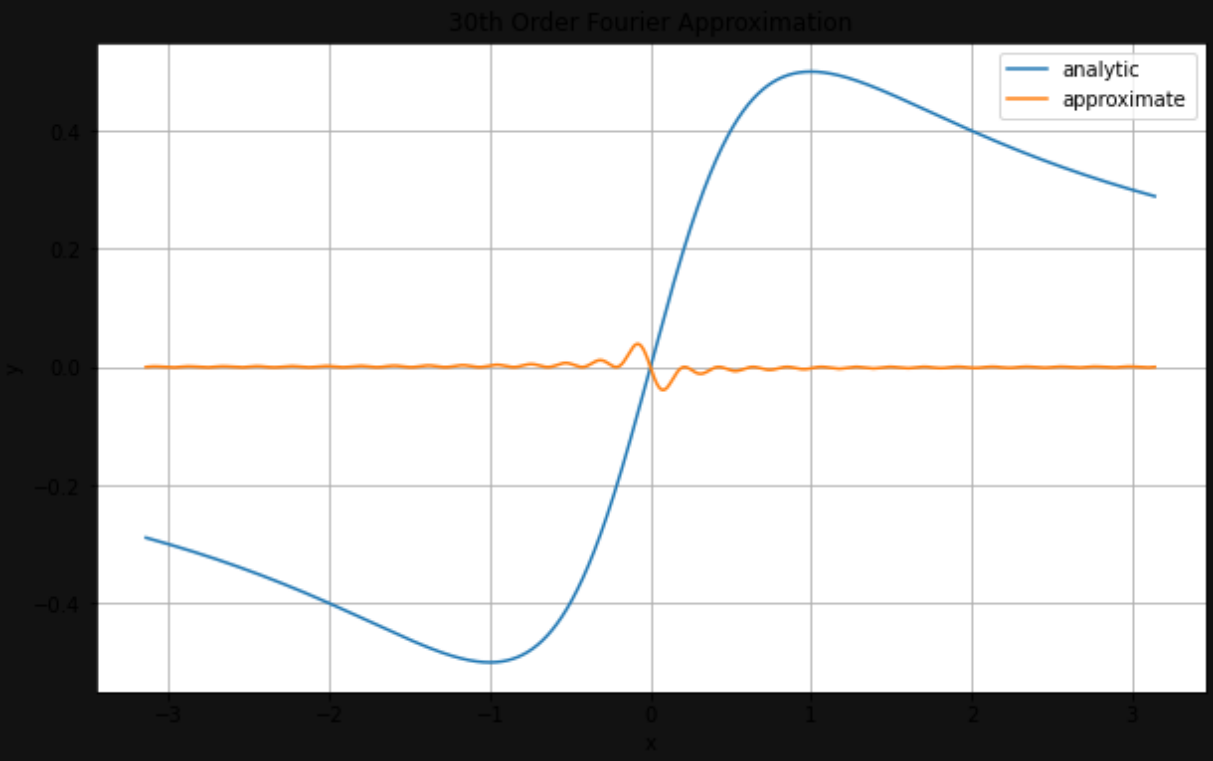
In [48]:
```
f = lambda x : x / ((x**2) + 1)
N = 10
plot_results(f, N)
```

```
C:\Users\austi\AppData\Local\Temp/ipykernel_15132/1227324076.py:10: Integration
Warning: The maximum number of subdivisions (50) has been achieved.
  If increasing the limit yields no improvement it is advised to analyze
  the integrand in order to determine the difficulties.  If the position of a
  local difficulty can be determined (singularity, discontinuity) one will
  probably gain from splitting up the interval and calling the integrator
  on the subranges.  Perhaps a special-purpose integrator should be used.
  Bn = quad(func2, -np.pi, np.pi)[0]
```



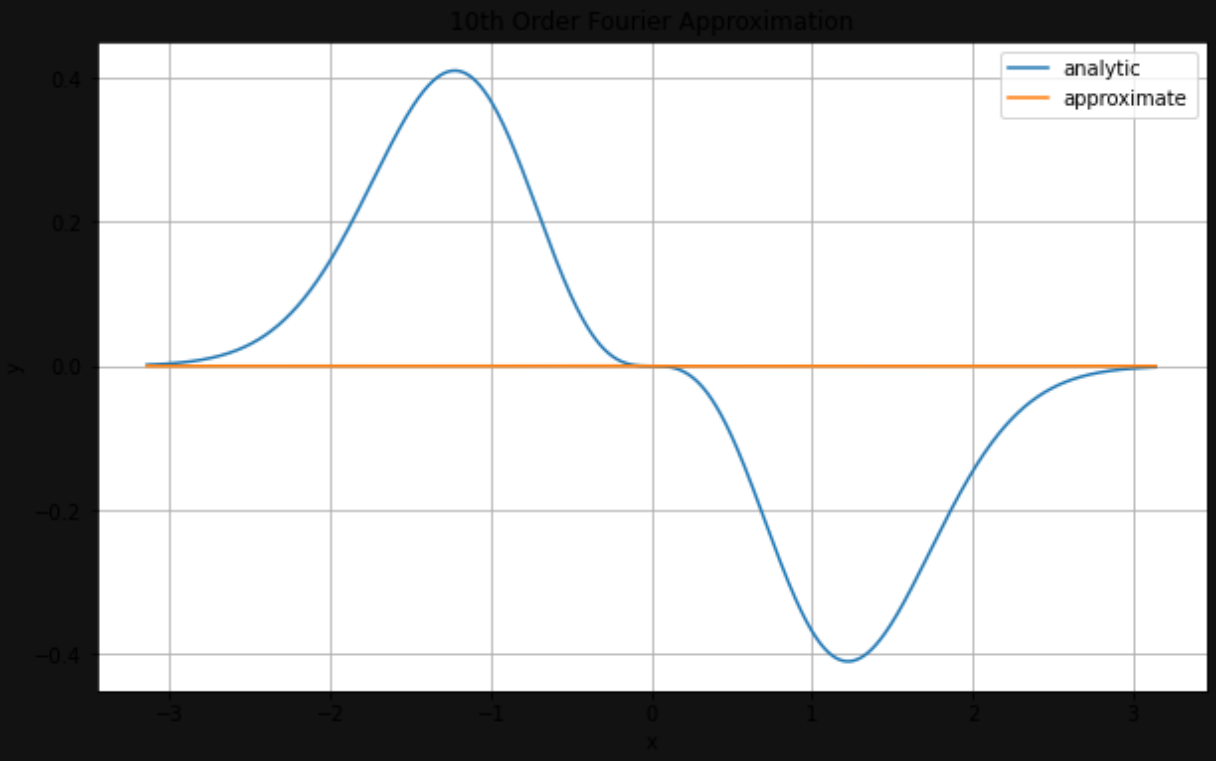
In [49]:
```
N = 30
plot_results(f, N)
```

```
C:\Users\austi\AppData\Local\Temp/ipykernel_15132/1227324076.py:10: Integration
Warning: The maximum number of subdivisions (50) has been achieved.
  If increasing the limit yields no improvement it is advised to analyze
  the integrand in order to determine the difficulties.  If the position of a
  local difficulty can be determined (singularity, discontinuity) one will
  probably gain from splitting up the interval and calling the integrator
  on the subranges.  Perhaps a special-purpose integrator should be used.
  Bn = quad(func2, -np.pi, np.pi)[0]
```



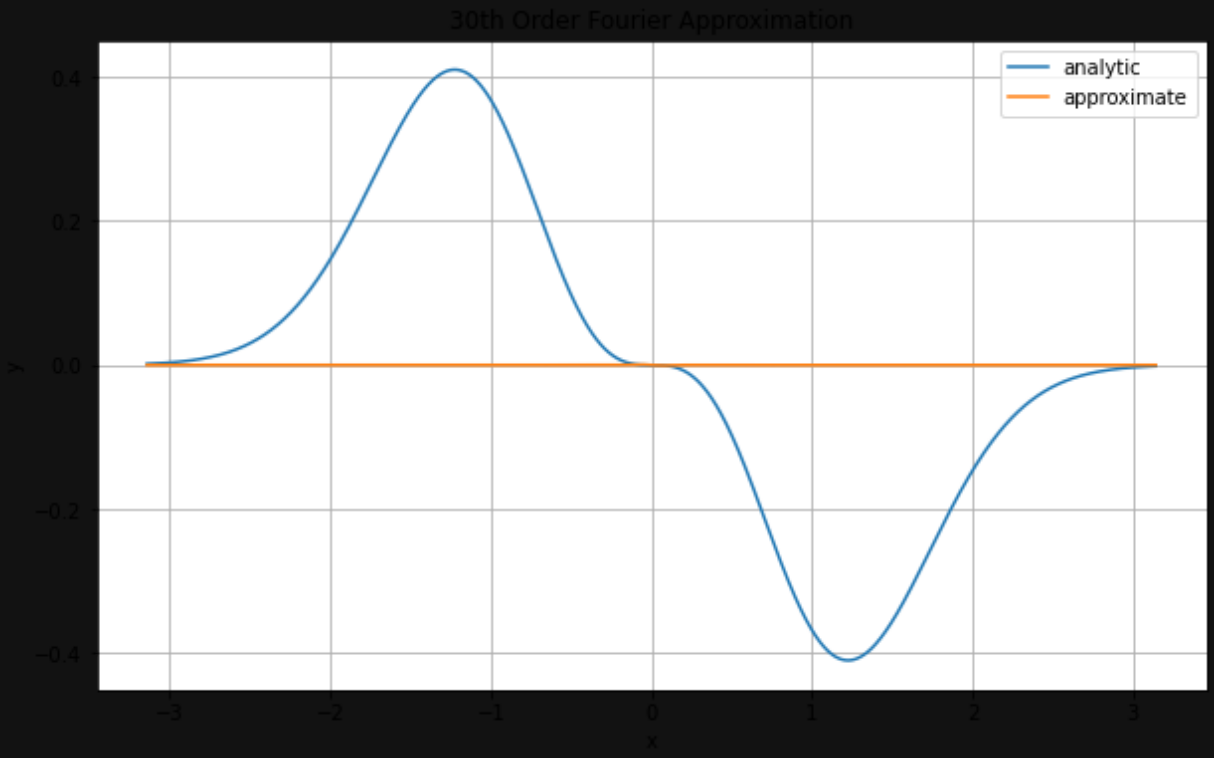
In [50]:
```
f = lambda x : (-x ** 3) * np.exp(-x ** 2)
N = 10
plot_results(f, N)
```

```
C:\Users\austi\AppData\Local\Temp/ipykernel_15132/1227324076.py:10: Integration
Warning: The integral is probably divergent, or slowly convergent.
  Bn = quad(func2, -np.pi, np.pi)[0]
```



In [51]:
```
N = 30
plot_results(f, N)
```

```
C:\Users\austi\AppData\Local\Temp/ipykernel_15132/1227324076.py:10: Integration
Warning: The integral is probably divergent, or slowly convergent.
  Bn = quad(func2, -np.pi, np.pi)[0]
```



In [ ]:
```
1
```