Homework 6

This week we will learned about numerical differentiation and numerical integration. We will be leaving the Differential Equations until Homework 7, where we delve into animation. All of these skills are extremely useful in any STEM based course. The goal for this homework is to apply what we learned for some problems and see how these tools can be used for our benefit.

In [1]:
```
#Code Preamble#
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
%matplotlib inline
```

Example: Numerical Differentiation

SciPy offers a library function to compute derivatives. It uses a central difference formula, but with additional ability to use more than two points. Here is an example of its use, it is called `scipy.mis.derivative` . See full documentation at http://docs.scipy.org/doc/scipy/reference/generated/scipy.misc.derivative.html

In [37]:
```
from scipy.misc import derivative
import time

# Compute the derivative of cos(x) at x = pi
t0 = time.time()
dy_scipy_1 = derivative(func=np.cos, x0=np.pi, dx=1e-6, n=1, order=3)
t1 = time.time()
dy_scipy_2 = derivative(func=np.cos, x0=np.pi, dx=1e-6, n=1, order=101)
t2 = time.time()

dt1 = t1 - t0
dt2 = t2 - t1
print("Derivative using order = 3: %.5f, takes %.5f sec to compute using your computer
print("Derivative using order = 101: %.5f, takes %.5f sec to compute using your comput
```

```
Derivative using order = 3: 0.00000, takes 0.00100 sec to compute using your computer
Derivative using order = 101: 0.00000, takes 0.00203 sec to compute using your comput
er
```

The arguments here are as follows:

```
func:  the name of the function whose derivative you want to
calculate
x0:    the location of the value (scalar) or values (numpy array)
where you want to evaluate the derivative
dx:    spacing between the points it generates (by evaluating `func`)
to calculate the differences
n:     the number of derivatives. `n=1` means first derivative of
```

```
`func`
order: number of points to use in the difference method. Must be odd.
```

Problem 1 (25 Points)

1. Use the `scipy.misc.derivative()` function with `order=3` to estimate the derivative of the $cos(x)$ and compute the difference relative to the analytical result (what you would get doing it by hand), and plot this difference. I give you a function to calculate the derivative from scratch using a centered method, plot the difference between this and the analytical result as well. Feel free to try changing some of the parameters like `dx` or `order` and see how they impact your result.

2. Try a different value of `n` (the number of derivatives). Plot your estimates of `n`-th derivative. Does it behave like you expect?

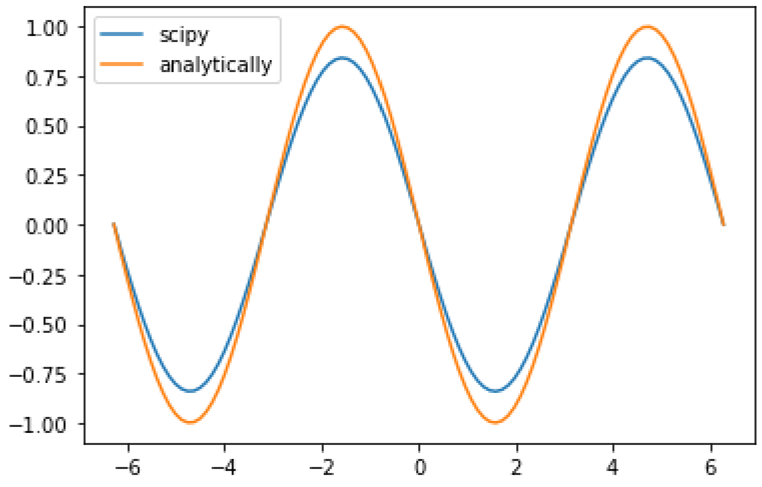
In [57]:
```
def centered_dy(y, x):
    '''
        Uses centered differences (see below) to estimate the derivatives at each valu
        except for the first and last values. The derivative at the first value of x i
        using a forward difference. The derivative at the last value of x is estimated
        using a backward difference.
            dy/dx at x[i] is approximated by (y[i+1] - y[i-1]) / (x[i+1]-x[i-1])
    '''
    dyc = [0.0]*len(x)
    dyc[0] = (y[0] - y[1])/(x[0] - x[1])
    for i in range(1,len(y)-1):
        dyc[i] = (y[i+1] - y[i-1])/(x[i+1]-x[i-1])
    dyc[-1] = (y[-1] - y[-2])/(x[-1] - x[-2])

    return dyc
```

In [78]:
```
#using Scipy
x = np.linspace(-2*np.pi,2*np.pi, 100)
cos_scipy_prime = derivative(func= np.cos, x0=x, dx=1, n=1, order=3)
plt.plot (x,cos_scipy_prime,label = "scipy")


#Analytically
cos_prime_analytic = -np.sin(x) #derivative of cos
plt.plot (x,cos_prime_analytic,label = "analytically")
plt.legend()
```

Out[78]:
```
<matplotlib.legend.Legend at 0x1856a2e3e20>
```

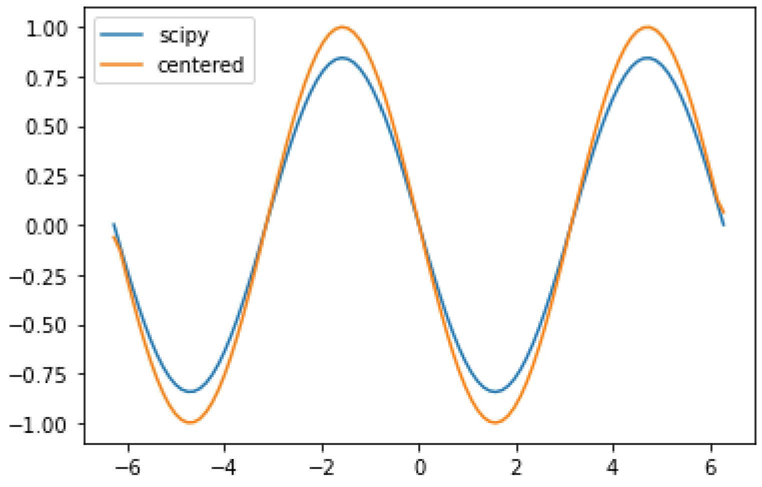


In [79]:
```
#using Scipy
x = np.linspace(-2*np.pi,2*np.pi, 100)
cos_scipy_prime = derivative(func= np.cos, x0=x, dx=1, n=1, order=3)
plt.plot (x,cos_scipy_prime,label = "scipy")

#using centered
f = lambda x: np.cos(x)
y = f(x)
cos_center_prime = centered_dy(y, x)
plt.plot (x,cos_center_prime,label = "centered")

plt.legend()
```

Out[79]:
```
<matplotlib.legend.Legend at 0x18569762a30>
```

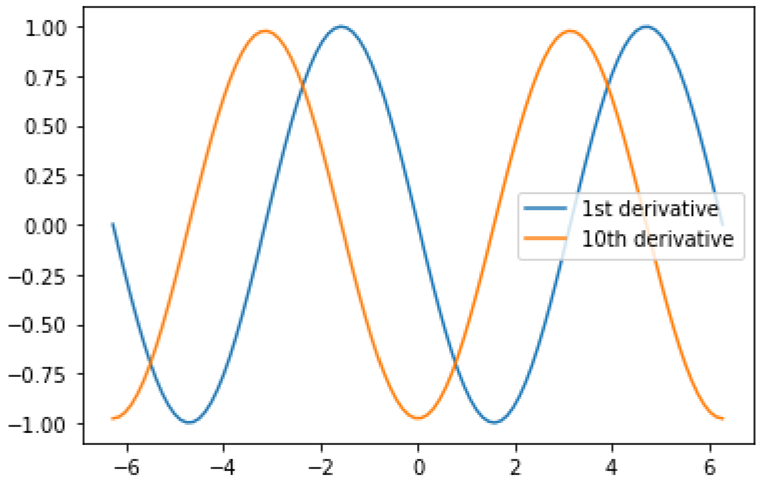


In [237…
```
#Different n
k = np.linspace(-2*np.pi,2*np.pi, 100)
cos_test_n_prime = derivative(func= np.cos, x0=k, dx=1, n=1, order=11)
plt.plot (x,cos_test_n_prime,label = "1st derivative")

cos_test_n_prime2 = derivative(func= np.cos, x0=x, dx=1, n=10, order=15)
plt.plot (x,cos_test_n_prime2,label = "10th derivative")
plt.legend()

#derivative order: cos -sin -cos sin cos
```

Out[237]:
```
<matplotlib.legend.Legend at 0x1857030ffa0>
```



I set n into 10 to make the 10th derivative and get a result I expected which is similar with the second derivative since they both have the same result which is -sin(x). However, I need to increase the order too since if I keep the same order as n=1, I will get a shorter graph.

Numerical Integration

Problem 2 (25 Points)

Adapted from [Ayars 2.2] and Physics 77

Compare results of the trapezoid integration method, Simpson's method, and the adaptive Gaussian quadrature method for the following integrals:

1. $$\int_0^{\pi} \sin x \, dx$$
2. $$\int_2^{4} (x^2 + x + 1) \, dx$$

For both parts, try it with more and with fewer slices to determine how many slices are required to give an 'acceptable' answer. (If you double the number of slices and still get the same answer, then try half as many, etc.) Part 2 is particularly interesting in this regard. In your submitted work, describe roughly how many points were required, and explain.

In [232…

```
from scipy import integrate
b = np.linspace(0,np.pi)
func = lambda b: np.sin(b)
sin_func = func(b)

#trapezoid method
trap = integrate.trapz(sin_func, x = None, dx = 0.06414, axis = 0)

#simpson method
simp = integrate.simps(sin_func, x = None, dx = 0.06414, axis = 0)
```

```
#quad method
quad = integrate.quad(func, 0, np.pi)


print("Using Trapezoid:", trap)
print("Using Simpson:", simp)
print("Using Quadrature:", quad)
```

```
Using Trapezoid: 2.0001213907113646
Using Simpson: 2.0008063013629585
Using Quadrature: (2.0, 2.220446049250313e-14)
```

In [230…

```
##### from scipy import integrate
c = np.linspace(2,4)
func2 = lambda c: c**2 + c + 1
function_c = func2(c)

#trapezoid method
trap2 = integrate.trapz(function_c, x = None, dx = 0.0408, axis = 0)

#simpson method
simp2 = integrate.simps(function_c, x = None, dx = 0.0408, axis = 0)

#quad method
quad2 = integrate.quad(func2, 2, 4)


print("Using Trapezoid:", trap2)
print("Using Simpson:", simp2)
print("Using Quadrature:", quad2)
```

```
Using Trapezoid: 26.656555102040816
Using Simpson: 26.656011328613076
Using Quadrature: (26.666666666666664, 2.9605947323337506e-13)
```

For the first integral, I need around 0.06 slices and for the second integral, I need 0.04 slices. I first tried to put the slice equal to 1 and it is far off from the real answer (using quadrature), and even more off if I increase it. I tried to decrease the slice, and the answer is getting smaller and closer to the answer. I did not expect it will be this low, but 0.06 and 0.04 is the slices that make the answer acceptable.