Imports

In [2]:

```python
import numpy as np
import matplotlib.pyplot as plt
import timeit
from scipy.misc import derivative
```

Differentiation

Differentiation of a function $f(x)$ is defined as:

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Differentiation Using Taylor Series

Taylor series gives a mean of approximationa function on a point $x + h$ (here denoted by x_{i+1}) as the value the function at x and its derivative at x. The Taylor series is:

$$f(x_{i+1}) = f(x_i) + \frac{f'(x_i)(x_{i+1} - x_i)}{1!} + \frac{f''(x_i)(x_{i+1} - x_i)^2}{2!} + \cdots + \frac{f^{(n)}(x_i)(x_{i+1} - x_i)^n}{n!} + \cdots$$

Using the above formula, the derivative of a function can be approximated.

Forwards Difference

First Order Derivative

Forward difference formula for the first order derivative is:

$$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} + O(h^2)$$

The above relation has an error of order h^2.

Second Order Derivative

The second order derivative is:

$$f''(x_i) = \frac{-f(x_{i+3}) + 4f(x_{i+2}) - 5f(x_{i+1}) + 2f(x_i)}{h^2} + O(h^2)$$

Again, the error is of order h^2.

Backwards Difference

First Order Derivative

First order derivative is:

$$f'(x_i) = \frac{3f(x_i) - 4f(x_{i-1}) + 3f(x_{i-2})}{2h} + O(h^2)$$

Second Order Derivative

The second order derivative is:

$$f''(x_i) = \frac{2f(x_i) - 5f(x_{i-1}) + 4f(x_{i-2}) - f(x_{i-3})}{h^2} + O(h^2)$$

Central Difference

The central difference formula can also be derived from Taylor expansion.

First Order Derivative

The first order derivative is:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2)$$

here the error is of order h^2.

We can also derive a formula with the error of order h^4. The formula is:

$$f'(x_i) = \frac{-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})}{12h} + O(h^4)$$

Second Order Derivative

A formula for second order derivative is with an error of $O(h^2)$ is:

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} + O(h^2)$$

While, for error of order h^4, the formula is:

$$f''(x_i) = \frac{-f(x_{i+2}) + 16f(x_{i+1}) - 30f(x_i) + 16f(x_{i-1}) - f(x_{i-2})}{12h^2} + O(h^4)$$

Implementation with Examples

We'll be using the function

$$f(x) = x^2 + sin(x)$$

At $x = 0.5$. The derivative of it, analytically is:

$$f'(x) = 2x + cos(x)$$

Which , for $x = 0.5$, is 1.8775825618903728.

Also, the second derivative is:

$$f''(x) = 2 - sin(x)$$

which, at $x = 0.5$, is 1.520574461395797.

In [3]:

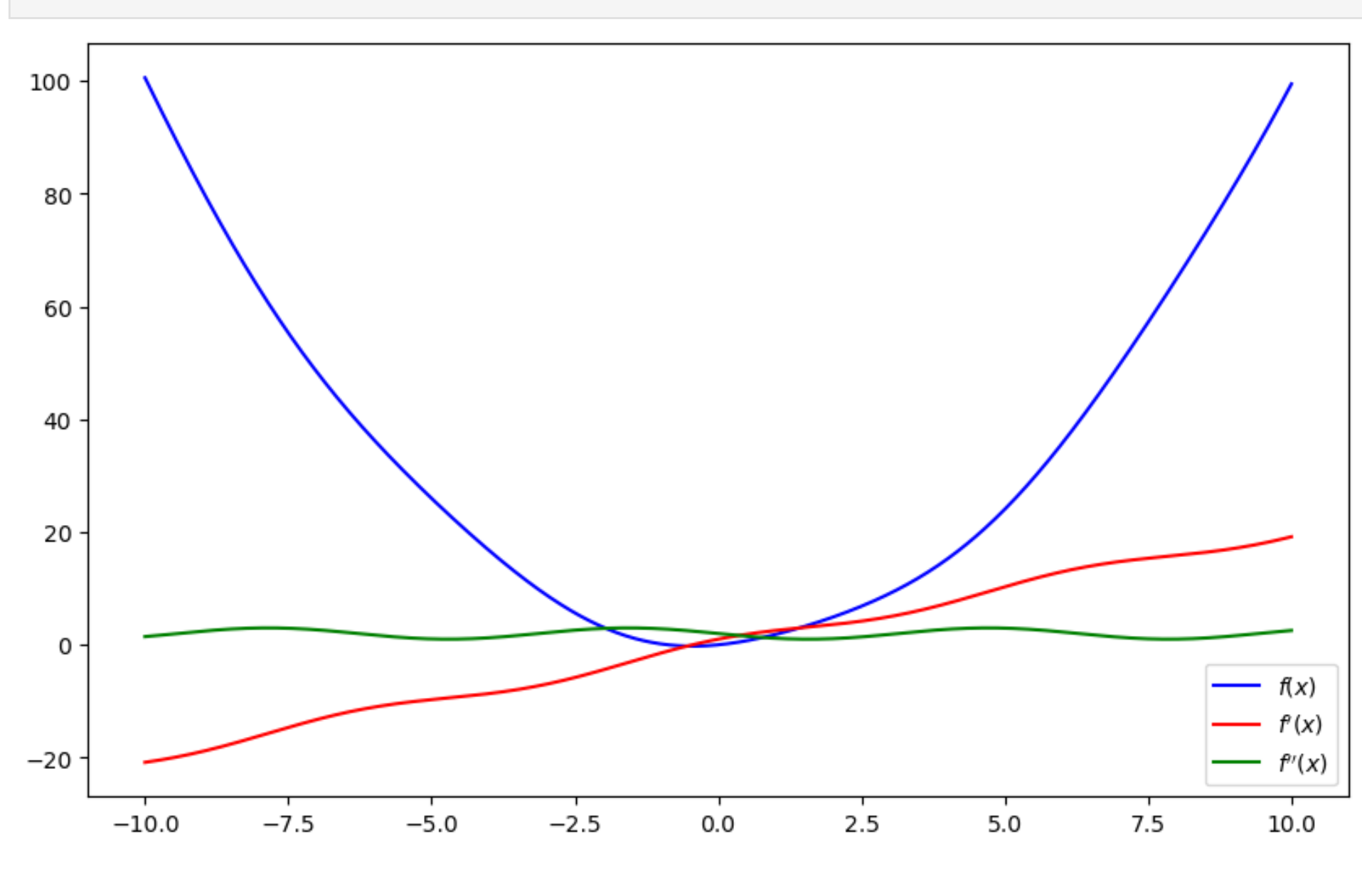
```python
def f(x):
    return x**2 + np.sin(x)

def fbarx(x):
    return 2*x + np.cos(x)

def fdoublebarx(x):
    return 2 - np.sin(x)
```

In [4]:

```python
plt.rcdefaults()
```

In [5]:

```python
X = np.linspace(-10, 10, 1000)
Y = f(X)
Ybar = fbarx(X)
Ydoublebar = fdoublebarx(X)
plt.figure(figsize=(10,6))
plt.plot(X, Y, label='$f(x)$', color='blue')
plt.plot(X, Ybar, label='$f\'(x)$', color='red')
plt.plot(X, Ydoublebar, label='$f\'\'(x)$', color='green')
plt.legend()
plt.show()
```



In [6]:

```python
#error
def error(y_calc, x=0.5, fdoublebar = False):
    if fdoublebar:
        y_true = fdoublebarx(x)
    else:
        y_true = fbarx(x)
    absolute_error = np.abs(y_true - y_calc)
    relative_error = absolute_error / y_true
    return absolute_error, relative_error
```

First Order Derivatives

Forward Difference

In [7]:

```python
#O(h) Implementaion
def oh(f, x, h=1e-3, report_error = True):
    y_calc = (f(x+h) - f(x))/(h)
    if report_error:
        ae, re = error(y_calc, x)
        print("Absolute Error:", ae)
        print("Relative Error:", re)
        return y_calc
    else:
        return y_calc
oh(f, 0.5)
```

```
Absolute Error: 0.0007601409868609466
Relative Error: 0.00040485089832514604
Out[7]: 1.8783427028772337
```

In [8]:

```python
#O(h2) Implementation of the Forward Difference
def fdoh2(f, x, h=1e-3, report_error = True):
    numerator = -f(x+2*h) + 4*f(x+h) - 3*f(x)
    denominator = 2*h
    y_calc = numerator/denominator
    if report_error:
        ae, re = error(y_calc, x)
        print("Absolute Error:", ae)
        print("Relative Error:", re)
        return y_calc
    else:
        return y_calc
```

In [9]:

```python
fdoh2(f, 0.5, h=1e-3)
```

```
Absolute Error: 2.9240747068115525e-07
Relative Error: 1.5573614530524607e-07
Out[9]: 1.8775828542978434
```

Backwards Difference

In [10]:

```python
#backward difference
def bdoh2(f, x, h=1e-3, report_error = True):
    numerator = 3*f(x) - 4*f(x-h) + f(x-2*h)
    denominator = 2*h
    y_calc = numerator/denominator
    if report_error:
        ae, re = error(y_calc, x)
        print("Absolute Error:", ae)
        print("Relative Error:", re)
        return y_calc
    else:
        return y_calc
bdoh2(f, 0.5, h=1e-3)
```

```
Absolute Error: 2.9264722334332305e-07
Relative Error: 1.558638375127868e-07
Out[10]: 1.877582854537596
```

Central Difference

In [11]:

```python
# central difference O(h2)
def cdoh2(f, x, h=1e-3, report_error = True):
    numerator = f(x+h) - f(x-h)
    denominator = 2*h
    y_calc = numerator/denominator
    if report_error:
        ae, re = error(y_calc, x)
        print("Absolute Error:", ae)
        print("Relative Error:", re)
        return y_calc
    else:
        return y_calc

cdoh2(f, 0.5, h=1e-3)
```

```
Absolute Error: 1.4626379529758538e-07
Relative Error: 7.790006057061226e-08
Out[11]: 1.8775824156265775
```

In [12]:

```python
#central difference O(h4)
def cdoh4(f, x, h=1e-3, report_error = True):
    numerator = -f(x+2*h) + 8*f(x+h) - 8*f(x-h) + f(x-2*h)
    denominator = 12*h
    y_calc = numerator/denominator
    if report_error:
        ae, re = error(y_calc, x)
        print("Absolute Error:", ae)
        print("Relative Error:", re)
        return y_calc
    else:
        return y_calc
cdoh4(f, 0.5, h=1e-3)
```

```
Absolute Error: 4.418687638008123e-14
Relative Error: 2.353391924112959e-14
Out[12]: 1.8775825618903286
```

Second Order Derivative

Forward Difference

In [13]:

```python
# forward difference O(h2)
def fdoh2second(f, x, h=1e-3, report_error = True):
    numerator = -f(x+3*h) + 4*f(x+2*h) - 5*f(x+h) + 2*f(x)
    denominator = h**2
    y_calc = numerator/denominator
    if report_error:
        ae, re = error(y_calc, x, fdoublebar=True)
        print("Absolute Error:", ae)
        print("Relative Error:", re)
        return y_calc
    else:
        return y_calc
fdoh2second(f, 0.5, h=1e-3)
```

```
Absolute Error: 4.400327222597866e-07
Relative Error: 2.8938584293719016e-07
Out[13]: 1.5205740213630747
```

Backwards Difference

In [14]:

```python
# backward difference O(h2) second order
def bdoh2second(f, x, h=1e-3, report_error = True):
    numerator = 2*f(x) - 5*f(x-h) +4*f(x-2*h) - f(x-3*h)
    denominator = h**2
    y_calc = numerator/denominator
    if report_error:
        ae, re = error(y_calc, x, fdoublebar=True)
        print("Absolute Error:", ae)
        print("Relative Error:", re)
        return y_calc
    else:
        return y_calc

bdoh2second(f, 0.5, h=1e-3)
```

```
Absolute Error: 4.3881147693269895e-07
Relative Error: 2.8858269560169784e-07
Out[14]: 1.52057402258432
```

Central Difference

In [15]:

```python
#central difference O(h2) second order
def cdoh2second(f, x, h=1e-3, report_error = True):
    numerator = f(x+h) - 2*f(x) + f(x-h)
    denominator = h**2
    y_calc = numerator/denominator
    if report_error:
        ae, re = error(y_calc, x, fdoublebar=True)
        print("Absolute Error:", ae)
        print("Relative Error:", re)
        return y_calc
    else:
        return y_calc
cdoh2second(f, 0.5, h=1e-3)
```

```
Absolute Error: 3.991669128566855e-08
Relative Error: 2.6251059911283398e-08
Out[15]: 1.5205745013124883
```

In [16]:

```python
#central difference O(h4) second order
def cdoh4second(f, x, h=1e-3, report_error = True):
    numerator = -f(x+2*h) + 16*f(x+h) - 30*f(x) + 16*f(x-h)- f(x-2*h)
    denominator = 12*h**2
    y_calc = numerator/denominator
    if report_error:
        ae, re = error(y_calc, x, fdoublebar=True)
        print("Absolute Error:", ae)
        print("Relative Error:", re)
        return y_calc
    else:
        return y_calc
cdoh4second(f, 0.5, h=1e-3)
```

```
Absolute Error: 1.4295142847231546e-10
Relative Error: 9.401146218192731e-11
Out[16]: 1.5205744615387484
```

Comparison of the Different Methods

The central difference is clearly more accurate than the forward and backwards difference. Let's see if it is faster too?

In [17]:

```python
oh_time = timeit.timeit(stmt="oh(f, 0.5, report_error=False)", number=100000, setup="
fdoh2_time = timeit.timeit(stmt="fdoh2(f, 0.5, report_error=False)", number=100000, se
bdoh2_time = timeit.timeit(stmt="bdoh2(f, 0.5, report_error=False)", number=100000, se
cdoh2_time = timeit.timeit(stmt="cdoh2(f, 0.5, report_error=False)", number=100000, se
cdoh4_time = timeit.timeit(stmt="cdoh4(f, 0.5, report_error=False)", number=100000, se
sp_time = timeit.timeit(stmt="derivative(f, 0.5, dx=1e-3)", number=100000, setup="fro
```

In [18]:

```python
oh_time, fdoh2_time, bdoh2_time, cdoh2_time, cdoh4_time, sp_time
```

```
Out[18]: (0.6321755000000167,
          1.0214599999999905,
          0.9381749000000354,
          0.6445823999999902,
          1.2573420999999598,
          4.404312899999979)
```

We can see that the central difference is indeed faster than the forward and backwards difference. In fact, central difference is faster than the inbuilt function in the `scipy` library.

In [19]:

```python
oh_time = timeit.timeit(stmt="oh(f, 0.5, report_error=False)", number=100000, setup="
fdoh2_time = timeit.timeit(stmt="fdoh2(f, 0.5, report_error=False)", number=100000, se
bdoh2_time = timeit.timeit(stmt="bdoh2(f, 0.5, report_error=False)", number=100000, se
cdoh2_time = timeit.timeit(stmt="cdoh2(f, 0.5, report_error=False)", number=100000, se
cdoh4_time = timeit.timeit(stmt="cdoh4(f, 0.5, report_error=False)", number=100000, se
sp_time = timeit.timeit(stmt="derivative(f, 0.5, dx=1e-3)", number=100000, setup="fro
oh_time, fdoh2_time, bdoh2_time, cdoh2_time, cdoh4_time, sp_time
```

```
Out[19]: (0.7947380999999609,
          1.382966500000009,
          1.4363558000000012,
          0.9500264999999786,
          1.7992227000000298,
          4.637720700000045)
```

In [20]:

```python
error(derivative(f, 0.5, dx=1e-3), 0.5), error(cdoh2(f, 0.5, report_error=False))
```

```
Out[20]: ((1.4626379529758538e-07, 7.790006057061226e-08),
          (1.4626379529758538e-07, 7.790006057061226e-08))
```

Error are the same in both the methods.

Other Methods

Richardson Extrapolation

Richardson extrapolation uses two derivative estimates to compute a third, more accurate approximation. For example, we can use two values for h to compute the derivative which is more accurate than the original estimate. For centered difference approximations with $O(h_2)$, the application of this formula will yield a new derivative estimate of $O(h_4)$. We use:

$$D = \frac{4}{3}D(h_1) - \frac{1}{3}D(h_2)$$

with $h_1 < h_2$. Let's see an example:

In [26]:

```python
cdoh2(f, 0.5,0.1)
cdoh2(f, 0.5,0.2)
```

```
Absolute Error: 0.0014619064584484587
Relative Error: 0.0007786110119049006
Absolute Error: 0.005838860449494332
Relative Error: 0.003109775606147352
Out[26]: 1.8717437014408784
```

In [25]:

```python
dh1 = cdoh2(f, 0.5, h=0.1, report_error=False)
dh2 = cdoh2(f, 0.5, h=0.2, report_error=False)
D = 4*dh1/3 - dh2/3
error(D, 0.5)
```

```
Out[25]: (2.921794766352903e-06, 1.5561471573378936e-06)
```

In [30]:

```python
dh1 = cdoh2(f, 0.5, h=0.01)
dh2 = cdoh2(f, 0.5, h=0.02)
```

```
Absolute Error: 1.4626302896392218e-05
Relative Error: 7.789965242149607e-06
Absolute Error: 5.850433402154742e-05
Relative Error: 3.115939357822143e-05
```

In [31]:

```python
dh1 = cdoh2(f, 0.5, h=0.02, report_error=False)
dh2 = cdoh2(f, 0.5, h=0.01, report_error=False)
D = 4*dh2/3 - dh1/3
error(D, 0.5)
```

```
Out[31]: (2.9252156252823625e-10, 1.5579691059429205e-10)
```

We can see that we get a better approximation of the derivative with Richardson extrapolation.

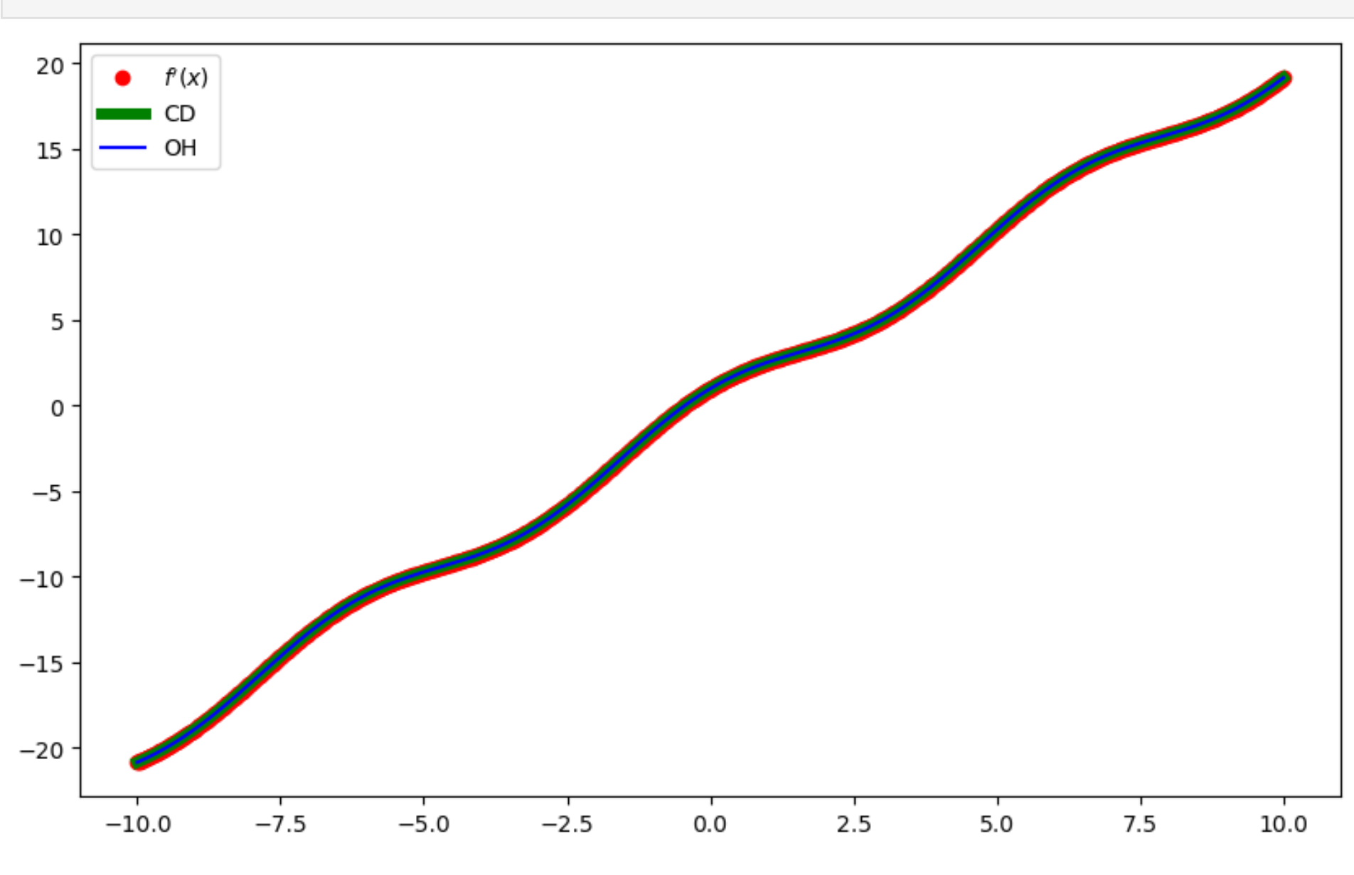
Finite Divided Difference Approximation

Remarks

The central difference formula is coming out to be the most accurate and time efficient.

In [53]:

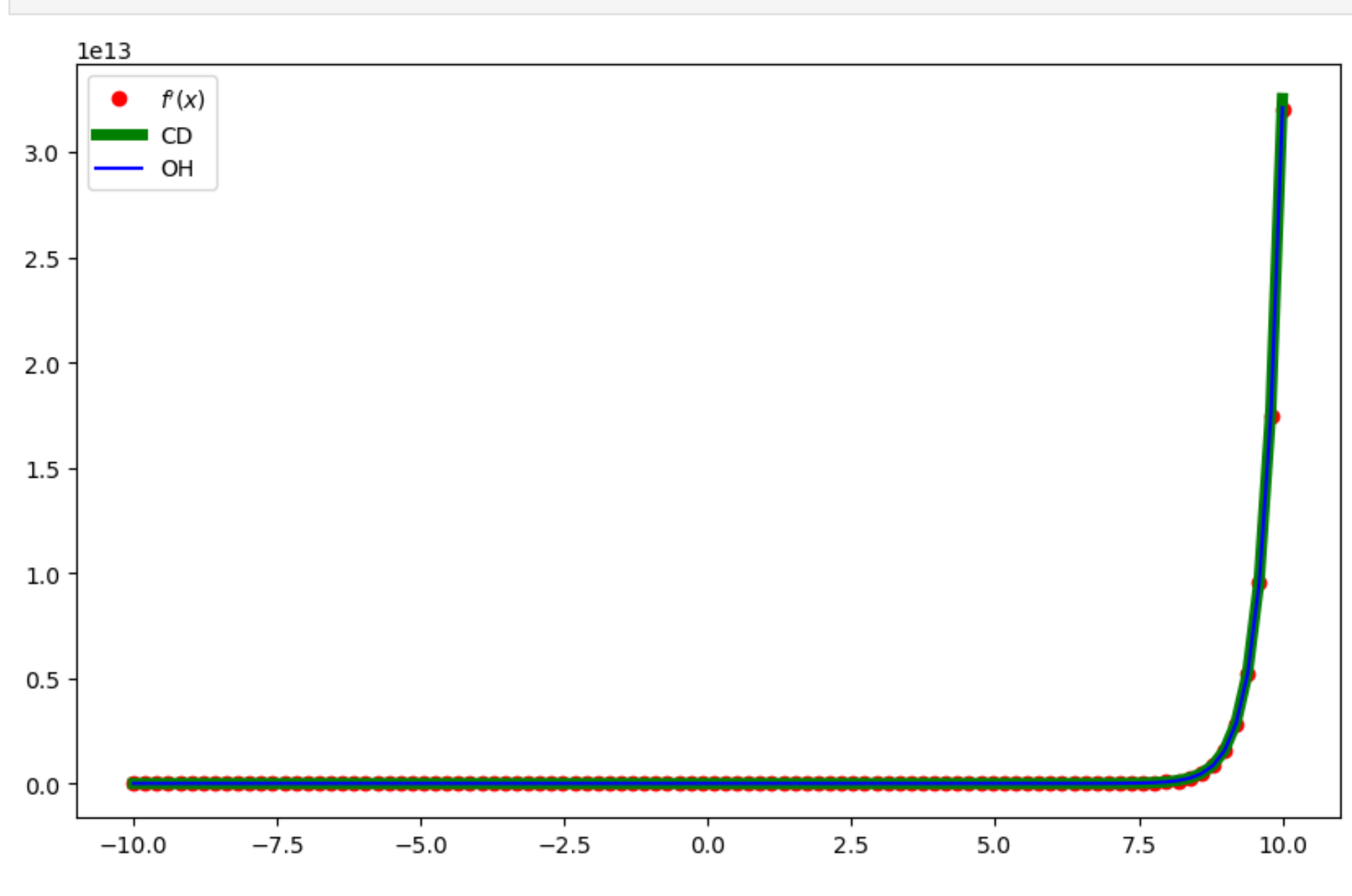
```python
def compare(f, fbar):
    X = np.linspace(-10, 10, 1000)
    Ybar = fbar(X)
    Ybar2 = cdoh2(f, X, h=0.1, report_error=False)
    Ybar3 = oh(f, X, report_error=False)
    plt.figure(figsize=(10,6))
    plt.plot(X, Ybar, "o" ,label='$f\'(x)$', color='red')
    plt.plot(X, Ybar2, label='CD', color='green', linewidth=5)
    plt.plot(X, Ybar3, label='OH', color='blue')
    plt.legend()
    plt.show()
compare(f, fbarx)
```



Let's use some more functions.

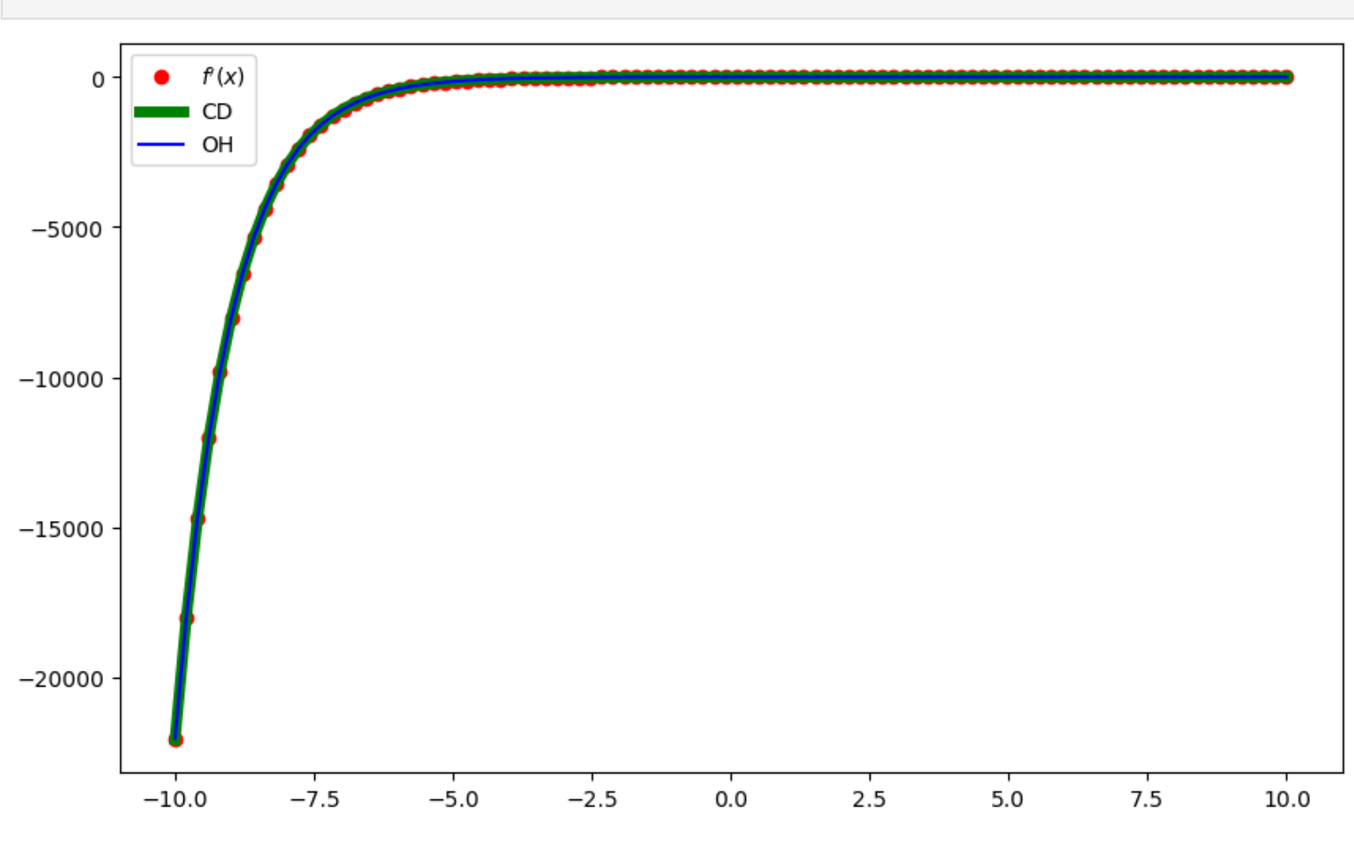
In [47]:

```python
def f2(x):
    return np.exp(3*x)
def f2bar(x):
    return 3*np.exp(3*x)
compare(f2, f2bar)
```



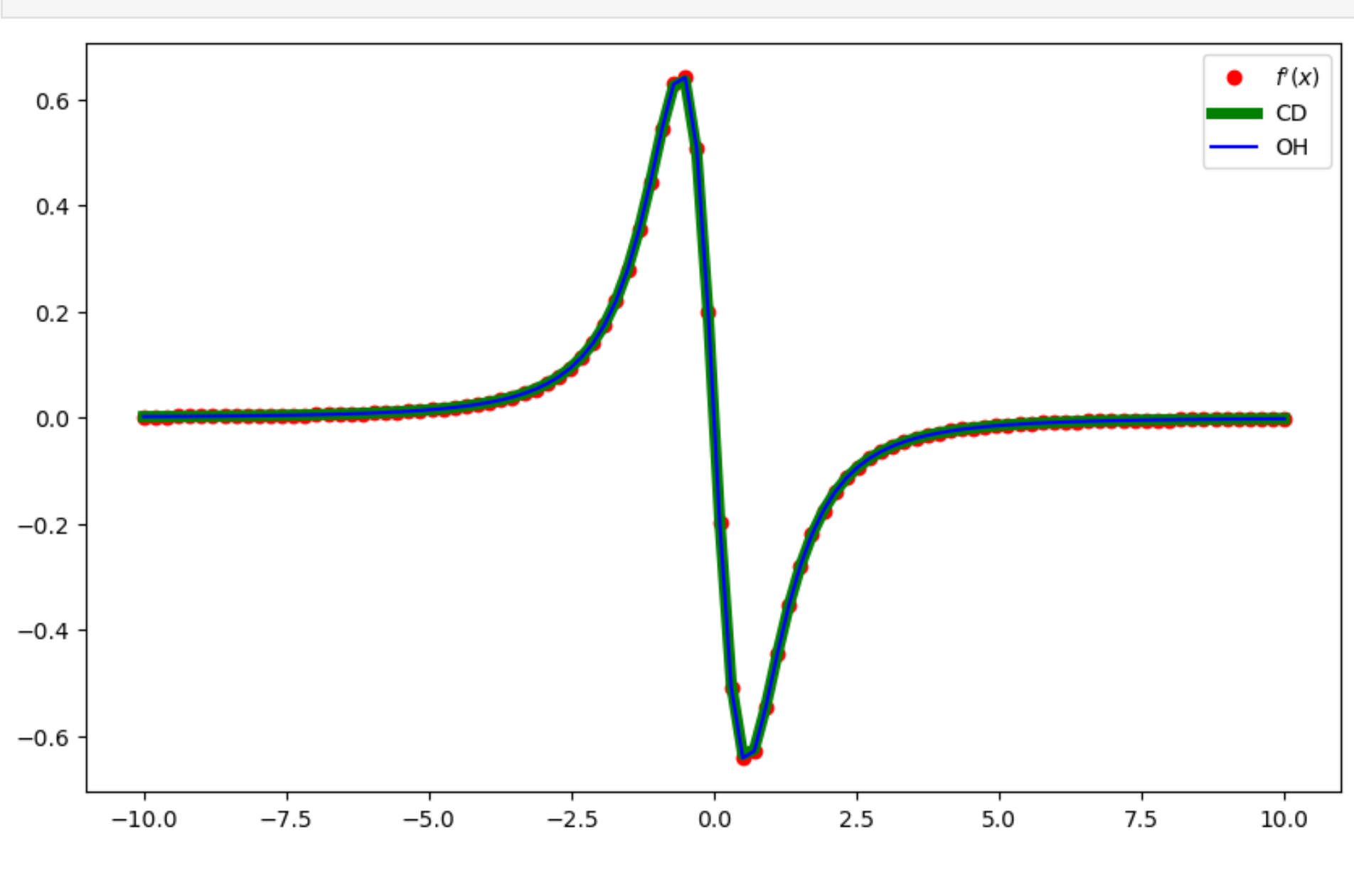
In [49]:

```python
f3 = lambda x: np.sin(x)+np.cos(x)+np.exp(-x)
f3bar = lambda x: np.cos(x)-np.sin(x)-np.exp(-x)
compare(f3, f3bar)
```



In [50]:

```python
f4 = lambda x: 1/(1+x**2)
f4bar = lambda x: -2*x/(1+x**2)**2
compare(f4, f4bar)
```



In [57]:

```python
a = 1
f4 = lambda x: np.sin(np.sqrt((np.exp(x)+a))/2)
f4bar = lambda x: ((np.exp(x))*(np.cos(np.sqrt((np.exp(x)+a))/2)))/(4*np.sqrt((np.exp
compare(f4, f4bar)
```