# IE 440 - NONLINEAR MODELS IN OPERATIONS RESEARCH

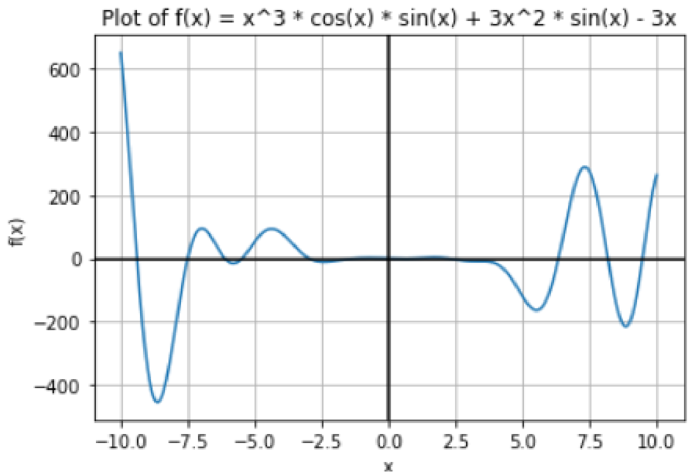
**Homework # 1**
**20.10.2023**

**Search Methods**

**TEAM OPTIMIZERS**
**Fatmanur Yaman - 2019402204**
**Ömercan Mısırlıoğlu - 2020402261**
**Hüseyin Emre Bacak - 2021402279**

**The Plot of The Function**

$$f(x) = x^3 \cos(x) \sin(x) + 3x^2 \sin(x) - 3x$$




**1. Bisection Method**

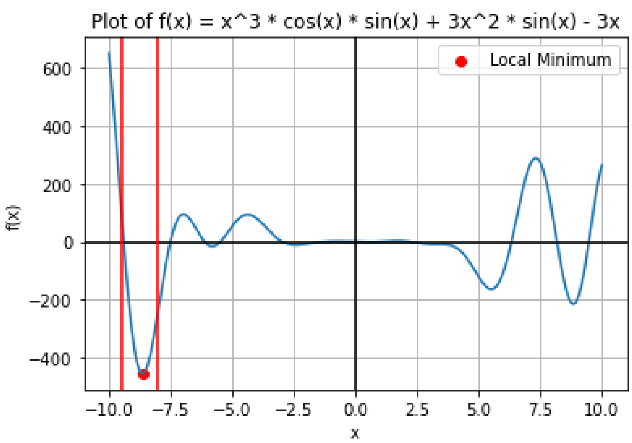
- **Parameters Used**

There are 3 parameters that the Bisection Method takes initially: *the left endpoint of the interval **a**, the right endpoint of the interval **b**, and the desired accuracy error **epsilon**.*

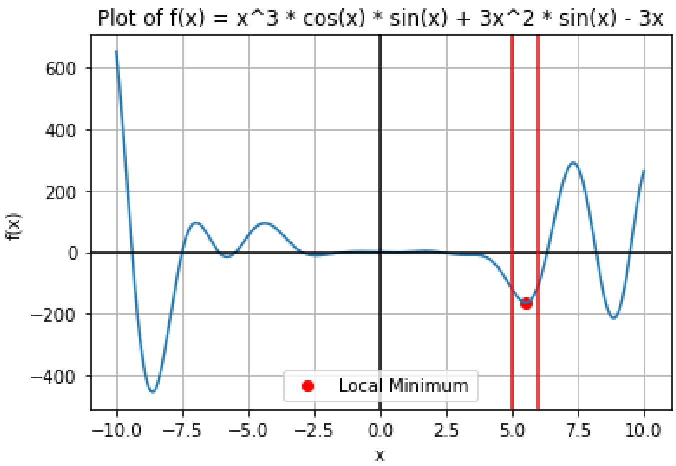
( a = 5, b = 6,  ε = 0.00001), ( a = 7.5, b = 10,  ε = 0.00001), ( a = -9.5, b = -8,  ε = 0.00001) cases were tried to see whether the algorithm converges to a local minimum or not.

- **Graph Of The Functions**

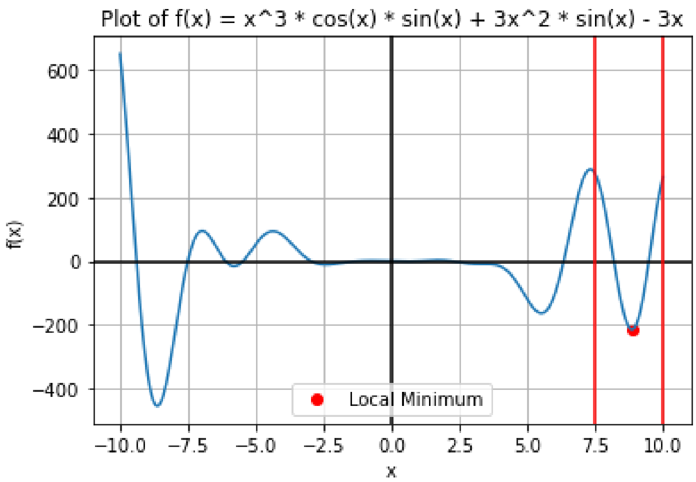
**Case 1:** ( a = -9.5, b = -8,  ε = 0.00001)



**Case 2:** ( a = 5, b = 6,  ε = 0.00001)




**Case 3:** ( a = 7.5, b = 10,  ε = 0.00001)



- **Results of the Code**

**Case 1:** ( a = -9.5, b = -8,  ε = 0.00001)

| | Iteration | a | b | x | f(x) | \|x(k+1) - x(k) / x(k) - x(k-1)\| | -log\|x(k+1) - x(k)\| + log\|x(k) - x(k-1)\| |
|---|---|---|---|---|---|---|---|
| 0 | 0 | -9.500000 | -8.000000 | -8.750000 | -444.037885 | None | None |
| 1 | 1 | -8.750000 | -8.000000 | -8.375000 | -410.978954 | 0.5 | 0.30103 |
| 2 | 2 | -8.750000 | -8.375000 | -8.562500 | -451.508563 | 0.5 | 0.30103 |
| 3 | 3 | -8.750000 | -8.562500 | -8.656250 | -454.403848 | 0.5 | 0.30103 |
| 4 | 4 | -8.656250 | -8.562500 | -8.609375 | -454.548164 | 0.5 | 0.30103 |
| 5 | 5 | -8.656250 | -8.609375 | -8.632812 | -454.883120 | 0.5 | 0.30103 |
| 6 | 6 | -8.632812 | -8.609375 | -8.621094 | -454.816334 | 0.5 | 0.30103 |
| 7 | 7 | -8.632812 | -8.621094 | -8.626953 | -454.875039 | 0.5 | 0.30103 |
| 8 | 8 | -8.632812 | -8.626953 | -8.629883 | -454.885425 | 0.5 | 0.30103 |
| 9 | 9 | -8.632812 | -8.629883 | -8.631348 | -454.885861 | 0.5 | 0.30103 |
| 10 | 10 | -8.631348 | -8.629883 | -8.630615 | -454.886040 | 0.5 | 0.30103 |
| 11 | 11 | -8.631348 | -8.630615 | -8.630981 | -454.886049 | 0.5 | 0.30103 |
| 12 | 12 | -8.630981 | -8.630615 | -8.630798 | -454.886069 | 0.5 | 0.30103 |
| 13 | 13 | -8.630981 | -8.630798 | -8.630890 | -454.886066 | 0.5 | 0.30103 |
| 14 | 14 | -8.630890 | -8.630798 | -8.630844 | -454.886069 | 0.5 | 0.30103 |
| 15 | 15 | -8.630844 | -8.630798 | -8.630821 | -454.886070 | 0.5 | 0.30103 |
| 16 | 16 | -8.630844 | -8.630821 | -8.630833 | -454.886069 | 0.5 | 0.30103 |
| 17 | 17 | -8.630833 | -8.630821 | -8.630827 | -454.886070 | 0.5 | 0.30103 |

```
x* = -8.630826950073242
f(x*) = -454.8860695211224
```

**Case 2:** ( a = 5, b = 6,  ε = 0.00001)

| | Iteration | a | b | x | f(x) | \|x(k+1) - x(k) / x(k) - x(k-1)\| | -log\|x(k+1) - x(k)\| + log\|x(k) - x(k-1)\| |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 5.000000 | 6.000000 | 5.500000 | -163.714470 | None | None |
| 1 | 1 | 5.500000 | 6.000000 | 5.750000 | -150.880764 | 0.5 | 0.30103 |
| 2 | 2 | 5.500000 | 5.750000 | 5.625000 | -161.061541 | 0.5 | 0.30103 |
| 3 | 3 | 5.500000 | 5.625000 | 5.562500 | -163.278208 | 0.5 | 0.30103 |
| 4 | 4 | 5.500000 | 5.562500 | 5.531250 | -163.711375 | 0.5 | 0.30103 |
| 5 | 5 | 5.500000 | 5.531250 | 5.515625 | -163.765677 | 0.5 | 0.30103 |
| 6 | 6 | 5.500000 | 5.515625 | 5.507812 | -163.753133 | 0.5 | 0.30103 |
| 7 | 7 | 5.507812 | 5.515625 | 5.511719 | -163.762686 | 0.5 | 0.30103 |
| 8 | 8 | 5.511719 | 5.515625 | 5.513672 | -163.765004 | 0.5 | 0.30103 |
| 9 | 9 | 5.513672 | 5.515625 | 5.514648 | -163.765546 | 0.5 | 0.30103 |
| 10 | 10 | 5.514648 | 5.515625 | 5.515137 | -163.765663 | 0.5 | 0.30103 |
| 11 | 11 | 5.515137 | 5.515625 | 5.515381 | -163.765683 | 0.5 | 0.30103 |
| 12 | 12 | 5.515381 | 5.515625 | 5.515503 | -163.765683 | 0.5 | 0.30103 |
| 13 | 13 | 5.515381 | 5.515503 | 5.515442 | -163.765684 | 0.5 | 0.30103 |
| 14 | 14 | 5.515381 | 5.515442 | 5.515411 | -163.765684 | 0.5 | 0.30103 |
| 15 | 15 | 5.515411 | 5.515442 | 5.515427 | -163.765684 | 0.5 | 0.30103 |
| 16 | 16 | 5.515427 | 5.515442 | 5.515434 | -163.765684 | 0.5 | 0.30103 |

```
x* = 5.515434265136719
f(x*) = -163.76568400972977
```

**Case 3:** ( a = 7.5, b = 10,  ε = 0.00001)

| | Iteration | a | b | x | f(x) | \|x(k+1) - x(k) / x(k) - x(k-1)\| | -log\|x(k+1) - x(k)\| + log\|x(k) - x(k-1)\| |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 7.500000 | 10.000000 | 8.750000 | -209.555318 | None | None |
| **1** | 1 | 8.750000 | 10.000000 | 9.375000 | -55.953427 | 0.5 | 0.30103 |
| **2** | 2 | 8.750000 | 9.375000 | 9.062500 | -186.526634 | 0.5 | 0.30103 |
| **3** | 3 | 8.750000 | 9.062500 | 8.906250 | -212.880585 | 0.5 | 0.30103 |
| **4** | 4 | 8.750000 | 8.906250 | 8.828125 | -214.905766 | 0.5 | 0.30103 |
| **5** | 5 | 8.828125 | 8.906250 | 8.867188 | -214.826393 | 0.5 | 0.30103 |
| **6** | 6 | 8.828125 | 8.867188 | 8.847656 | -215.098491 | 0.5 | 0.30103 |
| **7** | 7 | 8.828125 | 8.847656 | 8.837891 | -215.060074 | 0.5 | 0.30103 |
| **8** | 8 | 8.837891 | 8.847656 | 8.842773 | -215.093791 | 0.5 | 0.30103 |
| **9** | 9 | 8.842773 | 8.847656 | 8.845215 | -215.099770 | 0.5 | 0.30103 |
| **10** | 10 | 8.845215 | 8.847656 | 8.846436 | -215.100038 | 0.5 | 0.30103 |
| **11** | 11 | 8.845215 | 8.846436 | 8.845825 | -215.100131 | 0.5 | 0.30103 |
| **12** | 12 | 8.845825 | 8.846436 | 8.846130 | -215.100142 | 0.5 | 0.30103 |
| **13** | 13 | 8.845825 | 8.846130 | 8.845978 | -215.100151 | 0.5 | 0.30103 |
| **14** | 14 | 8.845978 | 8.846130 | 8.846054 | -215.100150 | 0.5 | 0.30103 |
| **15** | 15 | 8.845978 | 8.846054 | 8.846016 | -215.100151 | 0.5 | 0.30103 |
| **16** | 16 | 8.845978 | 8.846016 | 8.845997 | -215.100151 | 0.5 | 0.30103 |
| **17** | 17 | 8.845997 | 8.846016 | 8.846006 | -215.100151 | 0.5 | 0.30103 |

```
x* = 8.846006393432617
f(x*) = -215.1001511009524
```

- **The Source Code**

The bisection function takes three parameters. First, it takes the middle point of two endpoints as *x* value. It compares the value of the given function at this middle point *f(x)* with the value of the function at *f(x + ε)*. If *f(x)* is bigger than *f(x + ε)* it updates *a* as the new *x*. Otherwise it updates *b* as the new *x*. At each iteration function checks the difference between updated values of *x*. If the difference is smaller than or equal to the ε value, it returns the local minimum value. If not, it repeats the same procedure.

```python
def bisection_method(a,b,epsilon):
    #Bisection
    count = 0
    output = {
        "Iteration": list(),
        "a": list(),
        "b": list(),
        "x": list(),
        "f(x)": list(),
        "|x(k+1) - x(k) / x(k) - x(k-1)|": list(),
        "-log|x(k+1) - x(k)| + log|x(k) - x(k-1)|": list()
    }
    temp = 0
    while b-a >= epsilon:
        output["Iteration"].append(count)
        output["a"].append(a)
        output["b"].append(b)
        x = (a+b)/2
        output["x"].append(x)
        output["f(x)"].append((x**3)*math.cos(x)*math.sin(x)+3*(x**2)*math.sin(x)-3*x)
        x_e = x + epsilon
        if (x**3)*math.cos(x)*math.sin(x)+3*(x**2)*math.sin(x)-3*x >= (x_e**3)*math.cos(x_e)*math.sin(x_e)+3*(x_e**2)*math.si
            a = x
        else:
            b = x
        next_v = (a+b)/2

        if count == 0:
            output["|x(k+1) - x(k) / x(k) - x(k-1)|"].append("None")
            output["-log|x(k+1) - x(k)| + log|x(k) - x(k-1)|"].append("None")
        else:
            output["|x(k+1) - x(k) / x(k) - x(k-1)|"].append(abs((next_v - x)/(x - temp)))
            output["-log|x(k+1) - x(k)| + log|x(k) - x(k-1)|"].append(round(-math.log(abs(next_v - x), 10) +
                                                                    math.log(abs(x - temp), 10), 5))

        count += 1
        if count != 0:
            temp = x

    return output
```

**2. Golden Section Method**

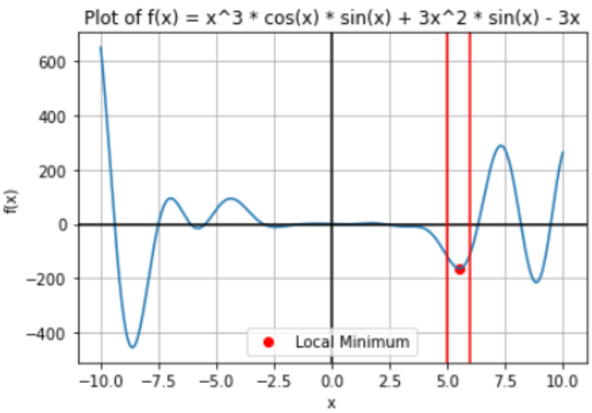
- **Parameters Used**

There are 3 parameters that the Golden Section Method takes initially: *the left endpoint of the interval **a**, the right endpoint of the interval **b**, and the desired accuracy error **epsilon**.*

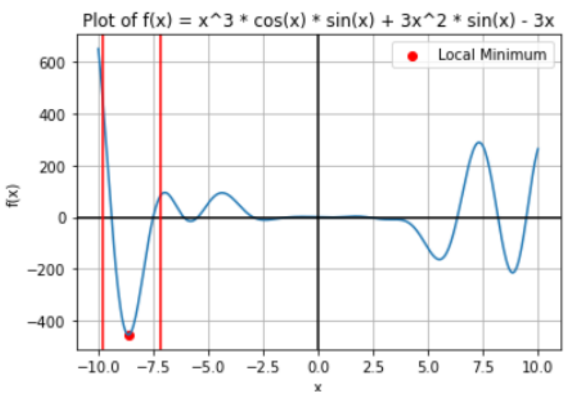
( a = 5, b = 6, ε = 0.00001), ( a = -9.8, b = -7.2, ε = 0.00001), ( a = 4, b = 7.1, ε = 0.00001) cases were tried to see whether the algorithm converges to a local minimum or not.

- **Graph Of The Functions**

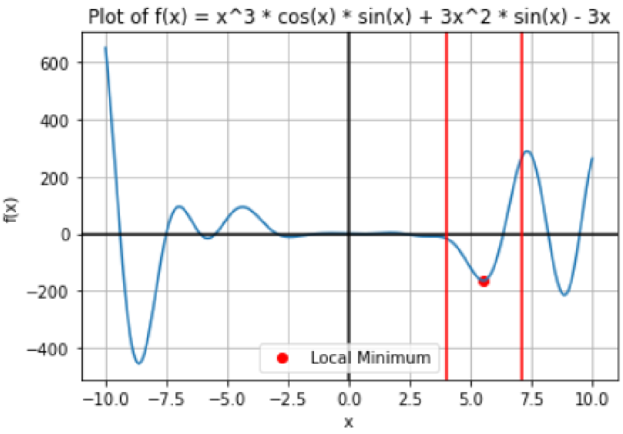
**Case 1:** ( a = 5, b = 6, ε = 0.00001)




**Case 2:** ( a = -9.8, b = -7.2, ε = 0.00001)



**Case 3:** ( a = 4, b = 7.1,  ε = 0.00001)




- **Results of the Code**

**Case 1:** ( a = 5, b = 6,  ε = 0.00001)

| | Iteration | a | b | x | y | f(x) | f(y) | \|x(k+1) - x(k) / x(k) - x(k-1)\| | -log\|x(k+1) - x(k)\| + log\|x(k) - x(k-1)\| |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 5.000000 | 6.000000 | 5.381953 | 5.618047 | -160.143476 | -161.399685 | None | None |
| **1** | 1 | 5.381953 | 6.000000 | 5.618047 | 5.763935 | -161.399685 | -149.250644 | 0.617801 | 0.20915 |
| **2** | 2 | 5.381953 | 5.763935 | 5.527852 | 5.618047 | -163.732264 | -161.399685 | 1.0 | 0.0 |
| **3** | 3 | 5.381953 | 5.618047 | 5.472130 | 5.527852 | -163.367784 | -163.732264 | 0.38183 | 0.41813 |
| **4** | 4 | 5.472130 | 5.618047 | 5.527852 | 5.562313 | -163.732264 | -163.282100 | 0.618644 | 0.20856 |
| **5** | 5 | 5.472130 | 5.562313 | 5.506576 | 5.527852 | -163.748746 | -163.732264 | 1.0 | 0.0 |
| **6** | 6 | 5.472130 | 5.527852 | 5.493413 | 5.506576 | -163.661773 | -163.748746 | 0.617082 | 0.20966 |
| **7** | 7 | 5.493413 | 5.527852 | 5.506576 | 5.514698 | -163.748746 | -163.765563 | 0.381631 | 0.41836 |
| **8** | 8 | 5.506576 | 5.527852 | 5.514698 | 5.519726 | -163.765563 | -163.761721 | 1.0 | 0.0 |
| **9** | 9 | 5.506576 | 5.519726 | 5.511598 | 5.514698 | -163.762490 | -163.765563 | 0.381061 | 0.419 |
| **10** | 10 | 5.511598 | 5.519726 | 5.514698 | 5.516621 | -163.765563 | -163.765386 | 1.0 | 0.0 |
| **11** | 11 | 5.511598 | 5.516621 | 5.513517 | 5.514698 | -163.764880 | -163.765563 | 0.624372 | 0.20456 |
| **12** | 12 | 5.513517 | 5.516621 | 5.514698 | 5.515436 | -163.765563 | -163.765684 | 0.384391 | 0.41523 |
| **13** | 13 | 5.514698 | 5.516621 | 5.515436 | 5.515887 | -163.765684 | -163.765642 | 1.0 | 0.0 |
| **14** | 14 | 5.514698 | 5.515887 | 5.515152 | 5.515436 | -163.765665 | -163.765684 | 0.388249 | 0.41089 |
| **15** | 15 | 5.515152 | 5.515887 | 5.515436 | 5.515606 | -163.765684 | -163.765679 | 1.0 | 0.0 |
| **16** | 16 | 5.515152 | 5.515606 | 5.515326 | 5.515436 | -163.765681 | -163.765684 | 0.398125 | 0.39998 |
| **17** | 17 | 5.515326 | 5.515606 | 5.515436 | 5.515499 | -163.765684 | -163.765683 | 1.0 | 0.0 |
| **18** | 18 | 5.515326 | 5.515499 | 5.515392 | 5.515436 | -163.765683 | -163.765684 | 0.422529 | 0.37414 |
| **19** | 19 | 5.515392 | 5.515499 | 5.515436 | 5.515458 | -163.765684 | -163.765684 | 1.0 | 0.0 |
| **20** | 20 | 5.515392 | 5.515458 | 5.515417 | 5.515436 | -163.765684 | -163.765684 | 0.366807 | 0.43556 |
| **21** | 21 | 5.515417 | 5.515458 | 5.515436 | 5.515442 | -163.765684 | -163.765684 | 1.04129 | -0.01757 |
| **22** | 22 | 5.515436 | 5.515458 | 5.515442 | 5.515449 | -163.765684 | -163.765684 | 0.477944 | 0.32062 |
| **23** | 23 | 5.515442 | 5.515458 | 5.515449 | 5.515452 | -163.765684 | -163.765684 | 0.293137 | 0.53293 |

```
x* = 5.51544949623405
f(x*) = -163.7656840411375
```

**Case 2:** ( a = -9.8, b = -7.2,  ε = 0.00001)

| | Iteration | a | b | x | y | f(x) | f(y) | \|x(k+1) - x(k) / x(k) - x(k-1)\| | -log\|x(k+1) - x(k)\| + log\|x(k) - x(k-1)\| |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | -9.800000 | -7.200000 | -8.806922 | -8.193078 | -430.916457 | -337.855692 | None | None |
| **1** | 1 | -9.800000 | -8.193078 | -9.186231 | -8.806922 | -210.245717 | -430.916457 | 0.618246 | 0.20884 |
| **2** | 2 | -9.186231 | -8.193078 | -8.806922 | -8.572416 | -430.916457 | -452.409466 | 0.382076 | 0.41785 |
| **3** | 3 | -8.806922 | -8.193078 | -8.572416 | -8.427538 | -452.409466 | -426.505627 | 0.617602 | 0.20929 |
| **4** | 4 | -8.806922 | -8.427538 | -8.662015 | -8.572416 | -454.159266 | -452.409466 | 1.0 | 0.0 |
| **5** | 5 | -8.806922 | -8.572416 | -8.717352 | -8.662015 | -449.211530 | -454.159266 | 0.618767 | 0.20847 |
| **6** | 6 | -8.717352 | -8.572416 | -8.662015 | -8.627775 | -454.159266 | -454.879229 | 0.382275 | 0.41762 |
| **7** | 7 | -8.662015 | -8.572416 | -8.627775 | -8.606639 | -454.879229 | -454.456782 | 1.0 | 0.0 |
| **8** | 8 | -8.662015 | -8.606639 | -8.640864 | -8.627775 | -454.811152 | -454.879229 | 0.382748 | 0.41709 |
| **9** | 9 | -8.640864 | -8.606639 | -8.627775 | -8.619711 | -454.879229 | -454.795127 | 0.614764 | 0.21129 |
| **10** | 10 | -8.640864 | -8.619711 | -8.632784 | -8.627775 | -454.883202 | -454.879229 | 1.0 | 0.0 |
| **11** | 11 | -8.640864 | -8.627775 | -8.635864 | -8.632784 | -454.867188 | -454.883202 | 0.623387 | 0.20524 |
| **12** | 12 | -8.635864 | -8.627775 | -8.632784 | -8.630864 | -454.883202 | -454.886068 | 0.384022 | 0.41564 |
| **13** | 13 | -8.632784 | -8.627775 | -8.630864 | -8.629688 | -454.886068 | -454.885128 | 1.0 | 0.0 |
| **14** | 14 | -8.632784 | -8.629688 | -8.631602 | -8.630864 | -454.885613 | -454.886068 | 0.387294 | 0.41196 |
| **15** | 15 | -8.631602 | -8.629688 | -8.630864 | -8.630419 | -454.886068 | -454.885953 | 1.0 | 0.0 |
| **16** | 16 | -8.631602 | -8.630419 | -8.631150 | -8.630864 | -454.885987 | -454.886068 | 0.395698 | 0.40264 |
| **17** | 17 | -8.631150 | -8.630419 | -8.630864 | -8.630698 | -454.886068 | -454.886059 | 1.0 | 0.0 |
| **18** | 18 | -8.631150 | -8.630698 | -8.630977 | -8.630864 | -454.886050 | -454.886068 | 0.527298 | 0.27794 |
| **19** | 19 | -8.630977 | -8.630698 | -8.630864 | -8.630805 | -454.886068 | -454.886070 | 0.341376 | 0.46677 |
| **20** | 20 | -8.630864 | -8.630698 | -8.630805 | -8.630762 | -454.886070 | -454.886067 | 0.810459 | 0.09127 |
| **21** | 21 | -8.630864 | -8.630762 | -8.630825 | -8.630805 | -454.886070 | -454.886070 | 1.0 | 0.0 |
| **22** | 22 | -8.630864 | -8.630805 | -8.630842 | -8.630825 | -454.886069 | -454.886070 | 0.380636 | 0.41949 |
| **23** | 23 | -8.630842 | -8.630805 | -8.630825 | -8.630819 | -454.886070 | -454.886070 | 0.234772 | 0.62935 |
| **24** | 24 | -8.630825 | -8.630805 | -8.630819 | -8.630813 | -454.886070 | -454.886070 | 1.0 | 0.0 |
| **25** | 25 | -8.630825 | -8.630813 | -8.630820 | -8.630819 | -454.886070 | -454.886070 | 2.259698 | -0.35405 |

```
x* = -8.630820396259566
f(x*) = -454.88606959122137
```

**Case 3:** ( a = 4, b = 7.1,  ε = 0.00001)

| | Iteration | a | b | x | y | f(x) | f(y) | \|x(k+1) - x(k) / x(k) - x(k-1)\| | -log\|x(k+1) - x(k)\| + log\|x(k) - x(k-1)\| |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 4.000000 | 7.100000 | 5.184054 | 5.915946 | -143.762801 | -124.827495 | None | None |
| **1** | 1 | 4.000000 | 5.915946 | 4.731801 | 5.184054 | -83.408686 | -143.762801 | 0.618246 | 0.20884 |
| **2** | 2 | 4.731801 | 5.915946 | 5.184054 | 5.463658 | -143.762801 | -163.199067 | 0.382076 | 0.41785 |
| **3** | 3 | 5.184054 | 5.915946 | 5.463658 | 5.636398 | -163.199067 | -160.456603 | 1.0 | 0.0 |
| **4** | 4 | 5.184054 | 5.636398 | 5.356828 | 5.463658 | -158.716081 | -163.199067 | 0.617403 | 0.20943 |
| **5** | 5 | 5.356828 | 5.636398 | 5.463658 | 5.529615 | -163.199067 | -163.722059 | 0.381754 | 0.41822 |
| **6** | 6 | 5.463658 | 5.636398 | 5.529615 | 5.570419 | -163.722059 | -163.098279 | 0.618967 | 0.20833 |
| **7** | 7 | 5.463658 | 5.570419 | 5.504436 | 5.529615 | -163.739612 | -163.722059 | 1.0 | 0.0 |
| **8** | 8 | 5.463658 | 5.529615 | 5.488851 | 5.504436 | -163.614580 | -163.739612 | 0.616561 | 0.21002 |
| **9** | 9 | 5.488851 | 5.529615 | 5.504436 | 5.514045 | -163.739612 | -163.765260 | 0.381431 | 0.41858 |
| **10** | 10 | 5.504436 | 5.529615 | 5.514045 | 5.519998 | -163.765260 | -163.761200 | 1.0 | 0.0 |
| **11** | 11 | 5.504436 | 5.519998 | 5.510380 | 5.514045 | -163.760149 | -163.765260 | 0.621829 | 0.20633 |
| **12** | 12 | 5.510380 | 5.519998 | 5.514045 | 5.516324 | -163.765260 | -163.765518 | 0.383436 | 0.41631 |
| **13** | 13 | 5.514045 | 5.519998 | 5.516324 | 5.517724 | -163.765518 | -163.764562 | 0.611866 | 0.21334 |
| **14** | 14 | 5.514045 | 5.517724 | 5.515450 | 5.516324 | -163.765684 | -163.765518 | 1.0 | 0.0 |
| **15** | 15 | 5.514045 | 5.516324 | 5.514916 | 5.515450 | -163.765623 | -163.765684 | 0.378124 | 0.42237 |
| **16** | 16 | 5.514916 | 5.516324 | 5.515450 | 5.515786 | -163.765684 | -163.765659 | 1.0 | 0.0 |
| **17** | 17 | 5.514916 | 5.515786 | 5.515248 | 5.515450 | -163.765676 | -163.765684 | 0.371779 | 0.42972 |
| **18** | 18 | 5.515248 | 5.515786 | 5.515450 | 5.515581 | -163.765684 | -163.765680 | 1.0 | 0.0 |
| **19** | 19 | 5.515248 | 5.515581 | 5.515375 | 5.515450 | -163.765683 | -163.765684 | 0.354537 | 0.45034 |
| **20** | 20 | 5.515375 | 5.515581 | 5.515450 | 5.515502 | -163.765684 | -163.765683 | 1.0 | 0.0 |
| **21** | 21 | 5.515375 | 5.515502 | 5.515424 | 5.515450 | -163.765684 | -163.765684 | 0.304571 | 0.51631 |
| **22** | 22 | 5.515424 | 5.515502 | 5.515450 | 5.515472 | -163.765684 | -163.765684 | 1.0 | 0.0 |
| **23** | 23 | 5.515424 | 5.515472 | 5.515442 | 5.515450 | -163.765684 | -163.765684 | 0.127819 | 0.8934 |
| **24** | 24 | 5.515442 | 5.515472 | 5.515450 | 5.515461 | -163.765684 | -163.765684 | 3.835317 | -0.5838 |
| **25** | 25 | 5.515442 | 5.515461 | 5.515449 | 5.515450 | -163.765684 | -163.765684 | 0.099588 | 1.00179 |

```
x* = 5.515449247975413
f(x*) = -163.76568404142915
```

- **The Source Code**

Golden section method takes 3 parameters. The parameters *a* and *b* denote the initial interval's first and last points respectively. The procedure in golden section method as follows;

First, we need to gather initial search points, say *x* & *y,* using *a* and *b* and *golden ratio*.

$$x = b - (1/\varphi)*(b-a)$$
$$y = a + (1/\varphi)*(b-a)$$

After that, we evaluate the function values of these points. While the difference between a and b values is tolerable (ε), we search for improvements in the evaluations.

If *f(x)* is greater than *f(y)*, we conclude that the local minimum lies in a smaller portion of the current interval. Also we know that the interval *[x,y)* is not important for us because of the unimodality assumption. So, we chop away the unnecessary parts of the interval. In order to do that, we assign the *x* value to the variable *a* (first point of an interval). After this assignment, we update the value of variable *x* as *y*'s value. The new calculation for new *y* is;

$$y = a + (1/\varphi)*(b-a)$$

After calculation, we evaluate the function for our brand new x and y. If *f(x)* is smaller than *f(y)*, a similar process happens as described above.

```python
def next_value(fx, fy, y0, a, b, g):
    if fx > fy:
        return y0
    else:
        return y0 - (1/g)*(y0-a)
```

```python
def golden_section(a,b,epsilon):
    g = 1.618     #golden ratio
    x0 = b - (1/g)*(b-a)
    y0 = a + (1/g)*(b-a)
    fx = x0**3 * math.cos(x0) * math.sin(x0) + 3 * x0**2 * math.sin(x0) - 3*x0
    fy = y0**3 * math.cos(y0) * math.sin(y0) + 3 * y0**2 * math.sin(y0) - 3*y0
    count = 0
    temp = 0
    next_v = x0
    output = {
        "Iteration": list(),
        "a": list(),
        "b": list(),
        "x": list(),
        "y": list(),
        "f(x)": list(),
        "f(y)": list(),
        "|x(k+1) - x(k) / x(k) - x(k-1)|": list(),
        "-log|x(k+1) - x(k)| + log|x(k) - x(k-1)|": list()
    }

    while b-a >= epsilon:
        output["Iteration"].append(count)
        output["a"].append(a)
        output["b"].append(b)
        output["x"].append(x0)
        output["y"].append(y0)
        output["f(x)"].append(fx)
        output["f(y)"].append(fy)
        if x0 > y0:
            x0, y0 = y0, x0

        if count != 0:
            temp = x0
```

```
        if fx > fy:
            a = x0
            x0 = y0
            y0 = a + (1/g)*(b-a)
            fx = fy
            fy = y0**3 * math.cos(y0) * math.sin(y0) + 3 * y0**2 * math.sin(y0) - 3*y0

        else:
            b = y0
            y0 = x0
            x0 = b - (1/g)*(b-a)
            fy = fx
            fx = x0**3 * math.cos(x0) * math.sin(x0) + 3 * x0**2 * math.sin(x0) - 3*x0

        next_v = next_value(fx, fy, y0, a, b, g)

        if count == 0:
            output["|x(k+1) - x(k) / x(k) - x(k-1)|"].append("None")
            output["-log|x(k+1) - x(k)| + log|x(k) - x(k-1)|"].append("None")
        else:
            output["|x(k+1) - x(k) / x(k) - x(k-1)|"].append(abs((next_v - x0)/(x0 - temp)))
            output["-log|x(k+1) - x(k)| + log|x(k) - x(k-1)|"].append(round(-math.log(abs(next_v - x0), 10) +
                                                                     math.log(abs(x0 - temp), 10), 5))

        count += 1
    return output
```

### 3. Newton's Method

### Algorithm Explanation

$$x = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})}$$

Input: $f,\ x^{(0)},\ \varepsilon$

Output: $x^{*},\ f'(x^{*}) = 0$

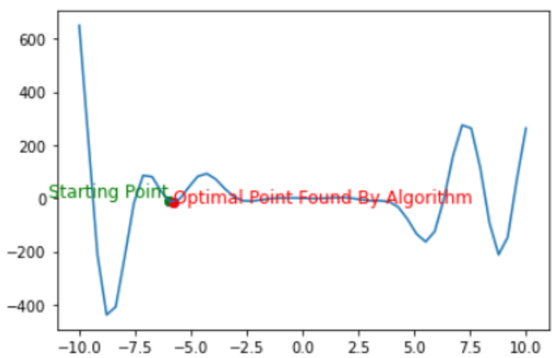
- **Parameters Used**

There are 2 parameters that Newton's Method takes initially: *initial point* $x_0$ *and the error* ε.

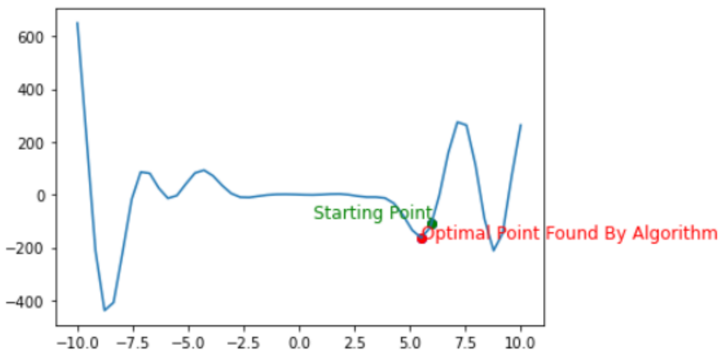
( $x_0 = 4$, $\varepsilon = 0.000000001$), ( $x_0 = 6$, $\varepsilon = 0.000000001$), ( $x_0 = 8$, $\varepsilon = 0.000000001$), ( $x_0 = 9$, $\varepsilon = 0.0005$) cases were tried to see whether the algorithm converges to a local minimum or not.

- **Graph Of The Functions**

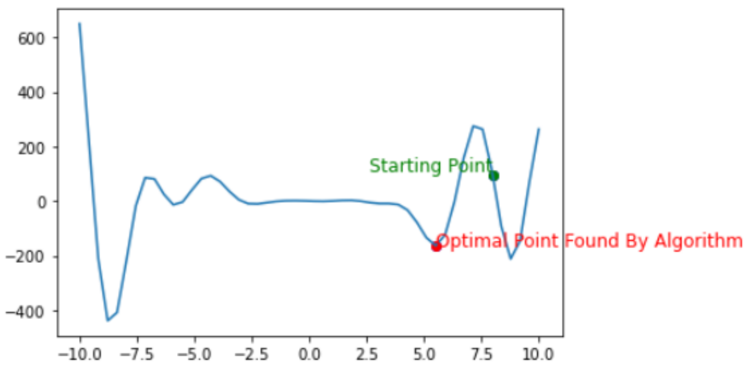
**Case 1:** ( $x_0 = 4$, $\varepsilon = 0.000000001$)



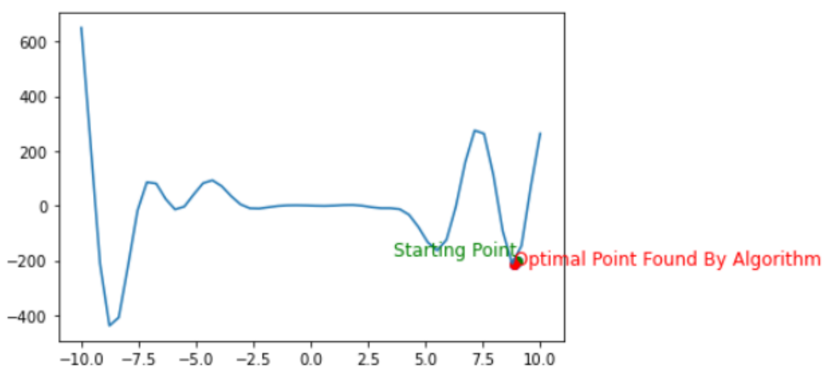
**Case 2:** ( $x_0 = 6$, $\varepsilon = 0.000000001$)




**Case 3:** ( $x_0 = 8$, $\varepsilon = 0.000000001$)




**Case 4:** ( $x_0 = 9$, $\varepsilon = 0.0005$)



- **Results of the Code**

**Case 1:** ( $x_0 = 4$, $\varepsilon = 0.000000001$)

| | Iteration | x_k | f_k(x_k) | f_k'(x_k) | f_k''(x_k) | Convergence |
|---|---|---|---|---|---|---|
| **0** | 0 | 4.000000 | -16.667056 | -38.105558 | -128.322749 | 1.596286 |
| **1** | 1 | 3.703049 | -10.126752 | -9.121672 | -64.802888 | 1.596286 |
| **2** | 2 | 3.562289 | -9.382562 | -2.161488 | -34.621175 | 3.151018 |
| **3** | 3 | 3.499856 | -9.307078 | -0.382301 | -22.544431 | 4.350550 |
| **4** | 4 | 3.482899 | -9.303687 | -0.026363 | -19.449254 | 4.713625 |
| **5** | 5 | 3.481543 | -9.303669 | -0.000165 | -19.205521 | 4.683111 |

```
x* = 3.4815431431137758
f(x*) = -9.30366946726433
```

**Case 2:** ( $x_0 = 6$, $\varepsilon = 0.000000001$)

| | Iteration | x_k | f_k(x_k) | f_k'(x_k) | f_k''(x_k) | Convergence |
|---|---|---|---|---|---|---|
| **0** | 0 | 6.000000 | -106.126749 | 243.936951 | 502.046285 | 0.005647 |
| **1** | 1 | 5.514115 | -163.765301 | -0.575085 | 431.374099 | 0.005647 |
| **2** | 2 | 5.515448 | -163.765684 | 0.000481 | 432.095359 | 0.626332 |

```
x* = 5.515447766160452
f(x*) = -163.76568404261627
```

**Case 3:** ( $x_0 = 8$, $\varepsilon = 0.000000001$)

| | Iteration | x_k | f_k(x_k) | f_k'(x_k) | f_k''(x_k) | Convergence |
|---|---|---|---|---|---|---|
| **0** | 0 | 8.000000 | 92.253534 | -501.407183 | -277.826561 | 0.261513 |
| **1** | 1 | 6.195251 | -49.499203 | 332.474156 | 390.329450 | 0.261513 |
| **2** | 2 | 5.343473 | -157.870676 | -65.494650 | 325.560959 | 0.277281 |
| **3** | 3 | 5.544648 | -163.579255 | 12.843106 | 447.375077 | 0.709335 |
| **4** | 4 | 5.515940 | -163.765631 | 0.213179 | 432.361086 | 0.598275 |
| **5** | 5 | 5.515447 | -163.765684 | 0.000066 | 432.094839 | 0.624746 |

```
x* = 5.515446804868879
f(x*) = -163.76568404287903
```

**Case 4:** ( $x_0 = 9$, $\varepsilon = 0.0005$)

| | Iteration | x_k | f_k(x_k) | f_k'(x_k) | f_k''(x_k) | Convergence |
|---|---|---|---|---|---|---|
| **0** | 0 | 9.000000 | -200.590060 | 187.975674 | 1199.502521 | 0.110669 |
| **1** | 1 | 8.843289 | -215.095657 | -3.308234 | 1217.217457 | 0.110669 |
| **2** | 2 | 8.846007 | -215.100151 | 0.001270 | 1218.145077 | 0.141151 |

```
x* = 8.8460065034334
f(x*) = -215.10015110082009
```

- **The Source Code**

The function called newtons_method takes two arguments as inputs: x and epsilon. Then, it gives a dataframe of results in each iteration. In the main function, there is another function called derivatives. The function simply takes the function and converts it to its first and second derivatives. By using these, the new points on the x-axis are being calculated according to Newton's algorithm that is mentioned earlier. The stopping conditions check the function also. The results in each iteration are being collected in a list so that it can be converted to a dataframe.

```python
def newtons_method(x, epsilon):

    df_list = []  #An empyt list for the results dataframe

    def derivatives(x_value):#A function that calculates the given function, first derivative, and the second derivative.
        #Taking the derivative of the function wrt X.
        x = symbols("x")
        f = (x**3)*cos(x)*sin(x)+3*x**2*sin(x)-3*x
        f_first = diff(f,x)
        f_second = diff(f_first,x)
        #Converting function to numerical type.
        numerical_f = lambdify(x, f, "math")
        numerical_f_first = lambdify(x, f_first, "math")
        numerical_f_second = lambdify(x, f_second, "math")
        #Calculating the functions with the given value.
        f_x = numerical_f(x_value)
        f_first_x = numerical_f_first(x_value)
        f_second_x = numerical_f_second(x_value)
        return [x_value, f_x, f_first_x, f_second_x] #Returning a list of x, f(x), f'(x), and f''(x).

    #Initial starting point of the algorithm.
    k = 0
    x = derivatives(x)[0]
    x_curr = x - (derivatives(x)[2]/derivatives(x)[3])
    x_fut = x_curr - (derivatives(x_curr)[2]/derivatives(x_curr)[3])
    convergence = abs(x_fut-x_curr)/(abs(x_curr-x)**2)
    row = [k, derivatives(x)[0], derivatives(x)[1],derivatives(x)[2],
           derivatives(x)[3],convergence]
    df_list.append(row)
    x_prev = x

    #As the stopping criteria is not being satisfied, the algorithms should work.
    while (abs(x_fut - x_curr)>epsilon and abs(derivatives(x_fut)[1] - derivatives(x_curr)[1])>epsilon
           and abs(derivatives(x_curr)[2])>epsilon):
        #The Newton's Method Algorithm
        x_curr = x_prev - (derivatives(x_prev)[2]/derivatives(x_prev)[3])
        x_fut = x_curr - (derivatives(x_curr)[2]/derivatives(x_curr)[3])
        convergence = abs(x_fut-x_curr)/(abs(x_curr-x_prev)**2)
        x_prev = x_curr
        k = k+1
         #Collecting the results.
         row = [k, derivatives(x_curr)[0], derivatives(x_curr)[1],derivatives(x_curr)[2],
                derivatives(x_curr)[3], convergence]
         df_list.append(row)
     return df_list #Returning dataframe of the results.
```

In addition to this, there are some codes to visualize the results.

```
column_names = ["Iteration", "x_k", "f_k(x_k)", "f_k'(x_k)", "f_k''(x_k)", "Convergence"]
df = pd.DataFrame(newtons_method(4, 0.000000001), columns=column_names)
df
```

```
print("x* = " + str(df.iloc[-1]["x_k"]))
print("f(x*) = " + str(df.iloc[-1]["f_k(x_k)"]))
```

```
def f(x):
    return x**3 * np.cos(x) * np.sin(x) + 3 * x**2 * np.sin(x) - 3 * x

x = np.linspace(-10, 10)
y = f(x)
plt.plot(x, y)
x1, y1 = df.iloc[0]["x_k"], df.iloc[0]["f_k(x_k)"]
x2, y2 = df.iloc[-1]["x_k"], df.iloc[-1]["f_k(x_k)"]
plt.scatter([x1, x2], [y1, y2], color='blue', label='Marked Points')
plt.scatter([x1], [y1], color='green', label='Marked Points')
plt.scatter([x2], [y2], color='red', label='Marked Points')
plt.text(x1, y1, 'Starting Point', fontsize=12, ha='right', va='bottom', color="green")
plt.text(x2, y2, 'Optimal Point Found By Algorithm', fontsize=12, ha='left', color="red")
```

## 4. Secant Method

### Algorithm Explanation

$$x^{(k+1)} = x^{(k)} - \frac{f^{'}(x^{(k)})}{f^{'}(x^{(k)}) - f^{'}(x^{(k-1)})} \times (x^{(k)} - x^{(k-1)})$$

Input: $f,\ x^{(0)},\ x^{(1)},\ \varepsilon$

Output: $x^{*},\ f^{'}(x^{*}) = 0$

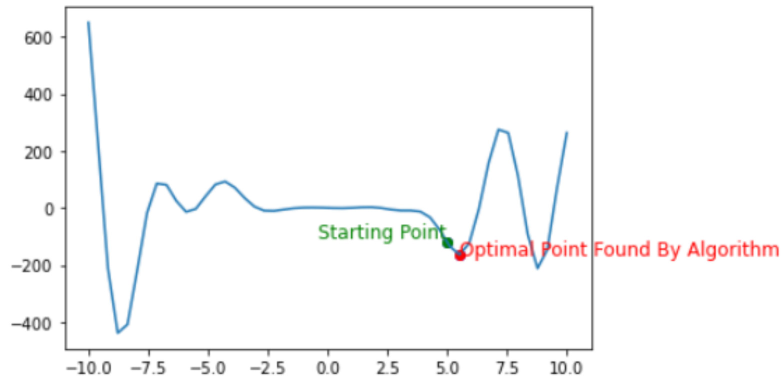
- **Parameters Used**

There are 3 parameters that Secant Method takes initially: *initial points* $x_0$ *and* $x_1$ *& the error* ε.

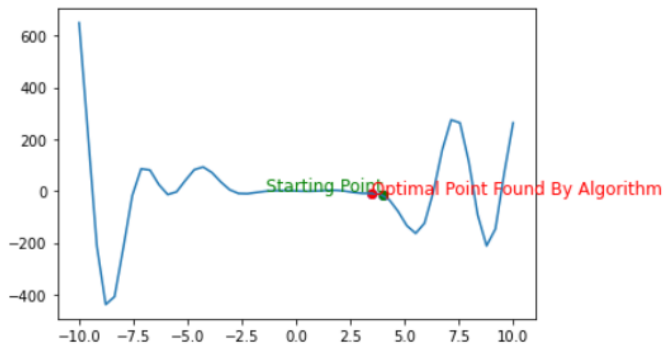
( $x_0$= 5, $x_1$= 7, ε = 0.0001), ( $x_0$= 4, $x_1$= 6, ε = 0.0001), ( $x_0$= 8, $x_1$= 10, ε = 0.0001), ( $x_0$= -6, $x_1$= -5, ε = 0.00005) cases were tried to see whether the algorithm converges to a local minimum or not.

- **Graph Of The Functions**

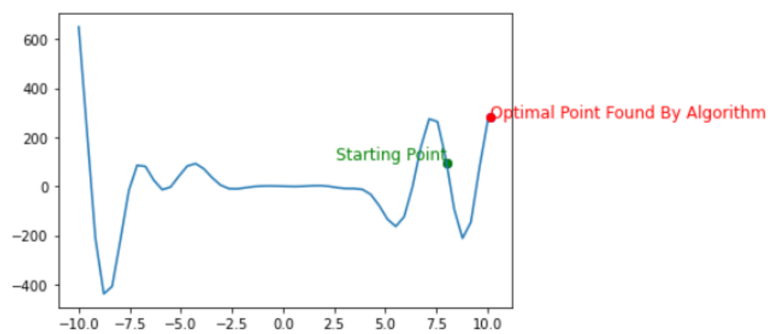
**Case 1:** ( $x_0 = 5$, $x_1 = 7$, $\varepsilon = 0.0001$)



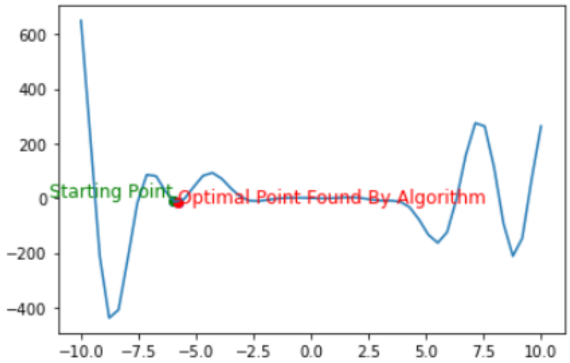

**Case 2:** ( $x_0 = 4$, $x_1 = 6$, $\varepsilon = 0.0001$)




**Case 3:** ( $x_0 = 8$, $x_1 = 10$, $\varepsilon = 0.0001$)



**Case 4:** ( $x_0$= -6, $x_1$= -5, ε = 0.00005)




- **Results of the Code**

**Case 1:** ( $x_0$= 5, $x_1$= 7, ε = 0.0001)

| | Iteration | x_k | f_k(x_k) | f_k'(x_k) | Convergence |
|---|---|---|---|---|---|
| **0** | 0 | 5.000000 | -120.920640 | -135.777797 | 0.425255 |
| **1** | 1 | 5.694684 | -156.360415 | 84.892431 | 0.422968 |
| **2** | 2 | 5.043752 | -126.770398 | -131.414774 | 0.792152 |
| **3** | 3 | 5.439217 | -162.551655 | -31.290219 | 0.554440 |
| **4** | 4 | 5.562806 | -163.271797 | 21.048703 | 1.464060 |
| **5** | 5 | 5.513103 | -163.764499 | -1.011102 | 0.292976 |
| **6** | 6 | 5.515381 | -163.765683 | -0.028237 | 1.234074 |

```
x* = 5.515381301919676
f(x*) = -163.76568312022212
```

**Case 2:** ( $x_0$= 4, $x_1$= 6, ε = 0.0001)

| | Iteration | x_k | f_k(x_k) | f_k'(x_k) | Convergence |
|---|---|---|---|---|---|
| **0** | 0 | 4.000000 | -16.667056 | -38.105558 | 0.563542 |
| **1** | 1 | 4.270211 | -32.138477 | -77.759043 | 0.172276 |
| **2** | 2 | 4.688329 | -77.508712 | -134.054477 | 4.081745 |
| **3** | 3 | 3.692680 | -10.035607 | -8.461711 | 0.067556 |
| **4** | 4 | 3.625598 | -9.597451 | -4.766080 | 6.849106 |
| **5** | 5 | 3.539087 | -9.341314 | -1.411911 | 1.910343 |
| **6** | 6 | 3.502671 | -9.308244 | -0.446490 | 3.582739 |
| **7** | 7 | 3.485829 | -9.303849 | -0.084133 | 2.896791 |
| **8** | 8 | 3.481919 | -9.303671 | -0.007391 | 2.962701 |
| **9** | 9 | 3.481542 | -9.303669 | -0.000145 | 2.621147 |

```
x* = 3.481542108116675
f(x*) = -9.303669467103587
```

**Case 3:** ( $x_0$= 8, $x_1$= 10, ε = 0.0001)

| | Iteration | x_k | f_k(x_k) | f_k'(x_k) | Convergence |
|---|---|---|---|---|---|
| 0 | 0 | 8.000000 | 92.253534 | -501.407183 | 0.221173 |
| 1 | 1 | 9.321112 | -84.344578 | 509.213410 | 2.571700 |
| 2 | 2 | 10.695374 | -14.302448 | -1078.325335 | 0.558083 |
| 3 | 3 | 9.761916 | 166.542990 | 523.760385 | 0.341136 |
| 4 | 4 | 10.067086 | 277.048952 | 151.214063 | 0.845211 |
| 5 | 5 | 10.190952 | 282.194185 | -73.551931 | 1.189647 |
| 6 | 6 | 10.150419 | 283.603833 | 3.496262 | 0.329011 |
| 7 | 7 | 10.152258 | 283.607111 | 0.066746 | 0.954180 |

```
x* = 10.15225783700372
f(x*) = 283.60711089335655
```

**Case 4:** ( $x_0$= -6, $x_1$= -5, ε = 0.00005)

| | Iteration | x_k | f_k(x_k) | f_k'(x_k) | Convergence |
|---|---|---|---|---|---|
| 0 | 0 | -6.000000 | -9.773001 | -62.658084 | 0.646897 |
| 1 | 1 | -5.646897 | -12.273999 | 46.622547 | 0.895184 |
| 2 | 2 | -6.089325 | -2.985381 | -88.885801 | 1.085763 |
| 3 | 3 | -5.799117 | -15.983351 | 1.116938 | 0.026657 |
| 4 | 4 | -5.802719 | -15.985326 | -0.020650 | 0.587520 |
| 5 | 5 | -5.802653 | -15.985327 | 0.000015 | 0.288899 |

```
x* = -5.802653396473716
f(x*) = -15.985326836442777
```

- **The Source Code**

The function called secant_method takes three arguments as inputs: x0, x1, and epsilon. Then, it gives a dataframe of results in each iteration. In the main function, there is another function called derivatives like in newtons_method. The function simply takes the function and converts it to its first derivatives since the Secant Method was designed to compute only the first derivative instead of the second derivative also. By using these, the new points on the x-axis are being calculated according to Secant's algorithm that is mentioned earlier. The stopping conditions check the function also. The results in each iteration are being collected in a list so that it can be converted to a dataframe.

```python
def secant_method(x0, x1, epsilon):

    df_list = []  #An empyt list for the results dataframe

    def derivatives(x_value): #Only the first derivative is needed.
        x = symbols("x")
        f = (x**3)*cos(x)*sin(x)+3*x**2*sin(x)-3*x
        f_first = diff(f,x)
        #Converting function to numerical type.
        numerical_f = lambdify(x, f, "math")
        numerical_f_first = lambdify(x, f_first, "math")
        #Calculating the functions with the given value.
        f_x = numerical_f(x_value)
        f_first_x = numerical_f_first(x_value)
        return [x_value, f_x, f_first_x] #Returning a list of x, f(x), and f'(x).

    k = 0
    x_prev = derivatives(x0)[0]
    x_curr = derivatives(x1)[0]
    x_fut = x_curr - (((derivatives(x_curr)[2]*(x_curr-x_prev))/(derivatives(x_curr)[2]-derivatives(x_prev)[2])))
    convergence = abs(x_fut-x_curr)/(abs(x_curr-x_prev)**1.618)
    row = [k, derivatives(x_prev)[0], derivatives(x_prev)[1],derivatives(x_prev)[2],convergence]
    x_prev = x1
    df_list.append(row)

    #As the stopping criteria is not being satisfied, the algorithms should work.
    while (abs(x_fut - x_curr)>epsilon):
        #The Secent Method Algorithm
        x_curr = x_fut
        x_fut = x_curr - ((derivatives(x_curr)[2]*(x_curr-x_prev))/(derivatives(x_curr)[2]-derivatives(x_prev)[2]))
        convergence = abs(x_fut-x_curr)/(abs(x_curr-x_prev)**1.618)
        x_prev = x_curr
        k = k+1
        #Collecting the results.
        row = [k, derivatives(x_prev)[0], derivatives(x_prev)[1],derivatives(x_prev)[2],convergence]
        df_list.append(row)
    return df_list #Returning dataframe of the results.
```

In addition to this, there are some codes to visualize the results.

```python
column_names = ["Iteration", "x_k", "f_k(x_k)", "f_k'(x_k)", "Convergence"]
df = pd.DataFrame(secant_method(5,7,0.0001), columns=column_names)
df
```

```python
print("x* = " + str(df.iloc[-1]["x_k"]))
print("f(x*) = " + str(df.iloc[-1]["f_k(x_k)"]))
```

```python
def f(x):
    return x**3 * np.cos(x) * np.sin(x) + 3 * x**2 * np.sin(x) - 3 * x

x = np.linspace(-10, 10)
y = f(x)
plt.plot(x, y)
x1, y1 = df.iloc[0]["x_k"], df.iloc[0]["f_k(x_k)"]
x2, y2 = df.iloc[-1]["x_k"], df.iloc[-1]["f_k(x_k)"]
plt.scatter([x1, x2], [y1, y2], color='blue', label='Marked Points')
plt.scatter([x1], [y1], color='green', label='Marked Points')
plt.scatter([x2], [y2], color='red', label='Marked Points')
plt.text(x1, y1, 'Starting Point', fontsize=12, ha='right', va='bottom', color="green")
plt.text(x2, y2, 'Optimal Point Found By Algorithm', fontsize=12, ha='left', color="red")
```

```python
        if fx > fy:
            a = x0
            x0 = y0
            y0 = a + (1/g)*(b-a)
            fx = fy
            fy = y0**3 * math.cos(y0) * math.sin(y0) + 3 * y0**2 * math.sin(y0) - 3*y0

        else:
            b = y0
            y0 = x0
            x0 = b - (1/g)*(b-a)
            fy = fx
            fx = x0**3 * math.cos(x0) * math.sin(x0) + 3 * x0**2 * math.sin(x0) - 3*x0

        next_v = next_value(fx, fy, y0, a, b, g)

        if count == 0:
            output["|x(k+1) - x(k) / x(k) - x(k-1)|"].append("None")
            output["-log|x(k+1) - x(k)| + log|x(k) - x(k-1)|"].append("None")
        else:
            output["|x(k+1) - x(k) / x(k) - x(k-1)|"].append(abs((next_v - x0)/(x0 - temp)))
            output["-log|x(k+1) - x(k)| + log|x(k) - x(k-1)|"].append(round(-math.log(abs(next_v - x0), 10) +
                                                                       math.log(abs(x0 - temp), 10), 5))


        count += 1
    return output
```