HW08 - Riemann Sums

November 4, 2024

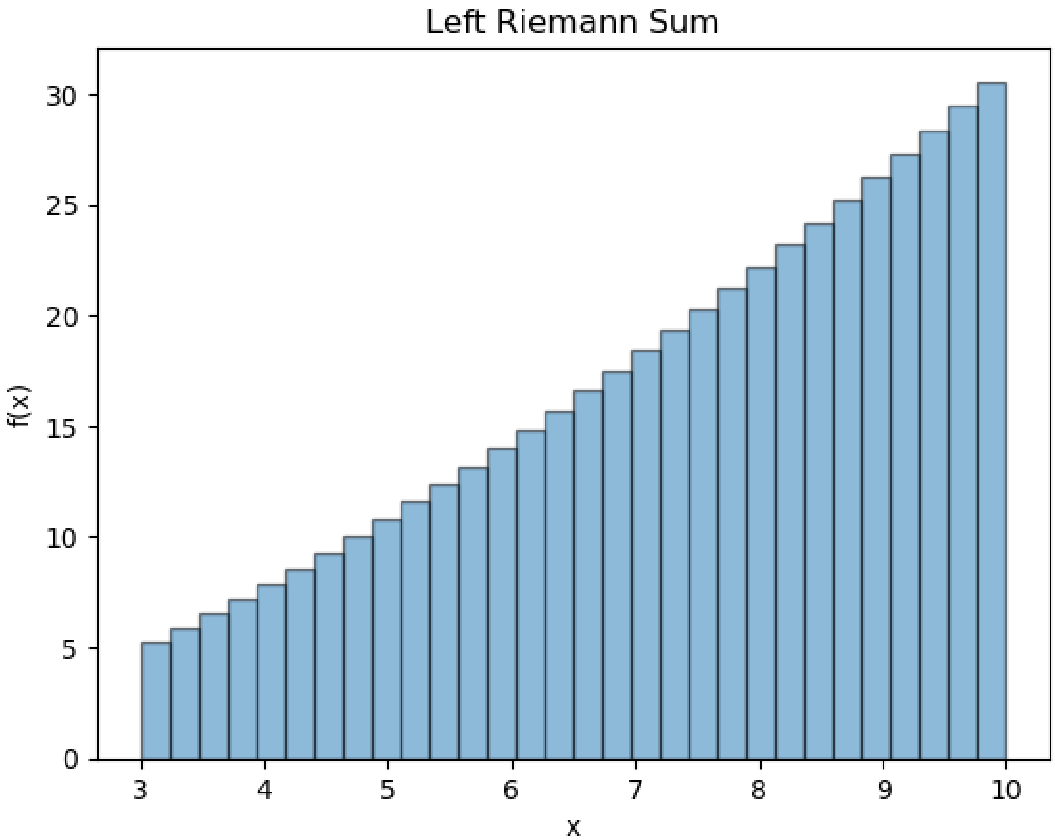
[10]:
```python
# Activity 1

import numpy as np
import matplotlib.pyplot as plt

def riemann(func, start, stop, steps, point, plot=False):
    f = func
    a = start
    b = stop
    dx = (b - a) / steps
    acc = 0
    heights = []
    x_vals = []

    if point == "left":
        x = a
        for k in range(steps):
            ds = f(x) * dx
            acc += ds
            heights.append(f(x))
            x_vals.append(x)
            x += dx
        print(ds, acc)

    elif point == "mid":
        x = a + dx / 2
        for k in range(steps):
            ds = f(x) * dx
            acc += ds
            heights.append(f(x))
            x_vals.append(x - dx/2)
            x += dx
        print(ds, acc)

    elif point == "right":
        x = a + dx
        for k in range(steps):
```

```
            ds = f(x) * dx
            acc += ds
            heights.append(f(x))
            x_vals.append(x - dx)
            x += dx
        print(ds, acc)

    if plot:
        plt.bar(x_vals, heights, width=dx, align='edge', edgecolor='black',␣
↪alpha=0.5)
        plt.xlabel('x')
        plt.ylabel('f(x)')
        plt.title(f'{point.capitalize()} Riemann Sum')
        plt.show()
```

[11]:
```
import math

g = lambda x: np.sqrt(1+x**3)
j = lambda x: math.sin(x)
c = lambda x: math.cos(x)
```

[12]:
```
riemann(g, 3, 10, 30, "left", plot=True)
```

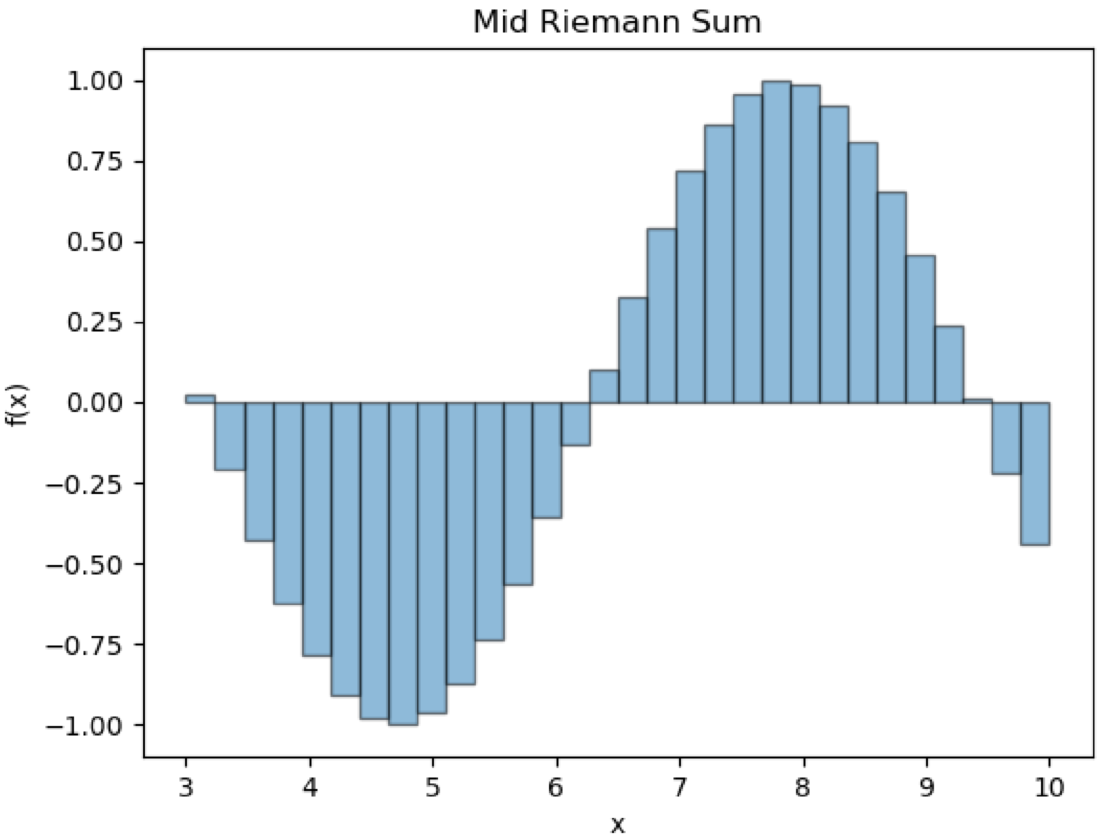
```
7.125728879041996 117.45220965709947
```



[13]:

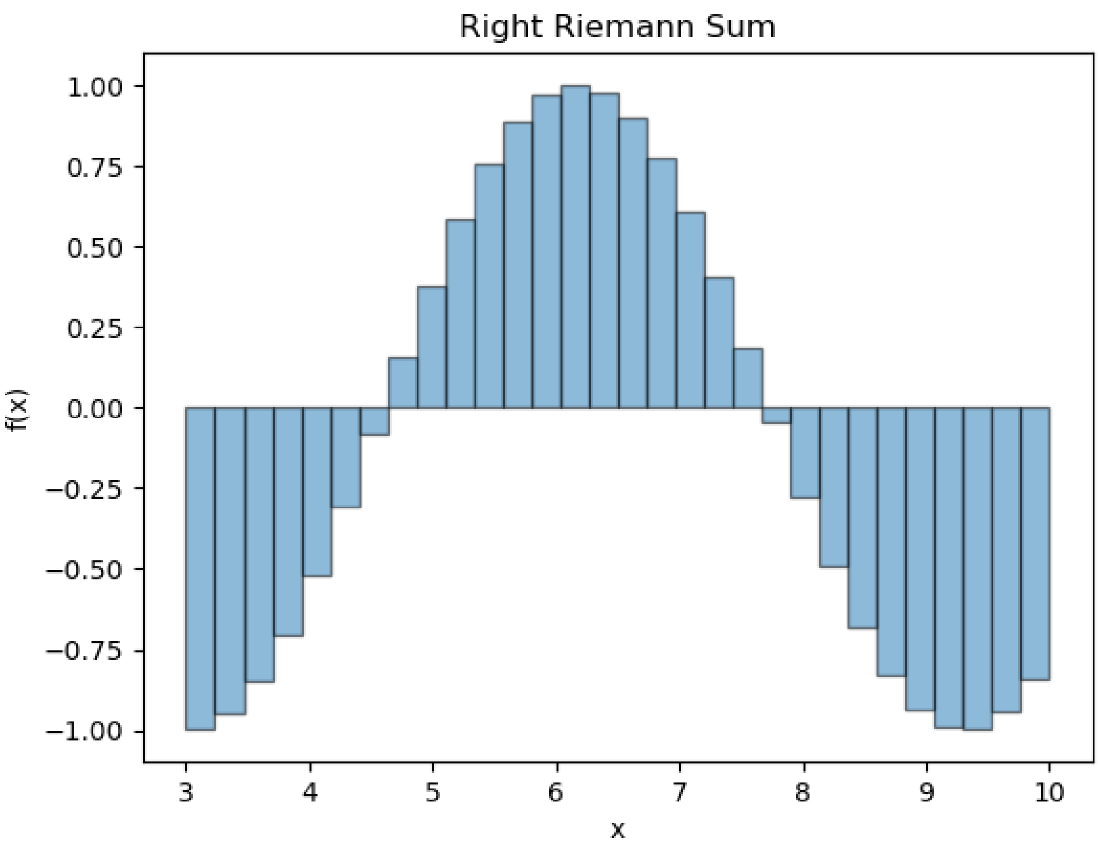
```
riemann(j, 3, 10, 30, "mid", plot=True)
```

```
-0.10328574250058763 -0.1512638789811302
```



[14]:
```
riemann(c, 3, 10, 30, "right", plot=True)
```

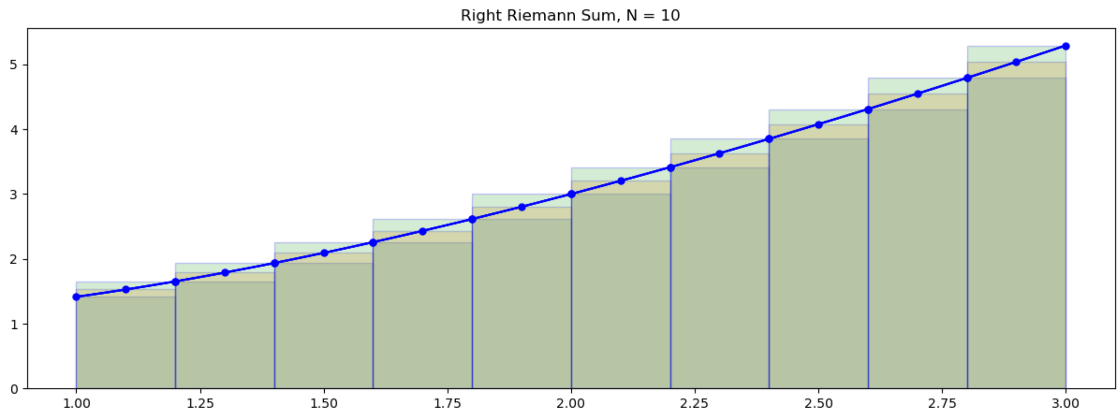
```
-0.19578335678450626 -0.6644223377588345
```



[15]:
```
# Activity 2 - using code from Eric Hernandez's piazza post

def three_point_riemann(func, start, stop, steps):

    f = func
    a = start
    b = stop
    N = steps
    n = 10 # Use n*N+1 points to plot the function smoothly

    x = np.linspace(a , b, N+1)
    y = f(x)

    X = np.linspace(a, b, n*N+1)
    Y = f(X)

    plt.figure(figsize=(15,5))

    #plt.subplot(1, 3, 1)
    plt.plot(X, Y, 'b')
```

```
    x_left = x[:-1] # Left endpoints
    y_left = y[:-1]
    plt.plot(x_left,y_left,'b.',markersize=10)
    plt.bar(x_left, y_left, width=(b - a) / N, alpha=0.2,␣
↪align='edge',edgecolor='b')
    plt.title('Left Riemann Sum, N = {}'.format(N))

    #plt.subplot(1, 3, 2)
    plt.plot(X, Y, 'b')
    x_mid = (x[:-1] + x[1:]) / 2 # Midpoints
    y_mid = f(x_mid)
    plt.plot(x_mid, y_mid, 'b.', markersize=10)
    plt.bar(x_mid, y_mid, width=(b - a) / N, alpha=0.2, edgecolor='b')
    plt.title('Midpoint Riemann Sum, N = {}'.format(N))

    #plt.subplot(1, 3, 3)
    plt.plot(X, Y, 'b')
    x_right = x[1:] # Left endpoints
    y_right = y[1:]
    plt.plot(x_right, y_right, 'b.', markersize=10)
    plt.bar(x_right ,y_right, width=-(b - a) / N, alpha=0.2, align='edge',␣
↪edgecolor='b')
    plt.title('Right Riemann Sum, N = {}'.format(N))

    plt.show()
```

[19]:
```
three_point_riemann(g, 1, 3, 10)
```



[20]:
```
# Activity 3
d = lambda x: math.sqrt(1 + x**3)

#4a)
```

```
riemann(d, 1, 3, 40, "left")
riemann(d, 1, 3, 400, "left")
riemann(d, 1, 3, 4000, "left")
riemann(d, 1, 3, 40000, "left")
# at 40000, it has stabilized at 3-4 decimal places

#4b) 6.2299
```

```
0.25822652361831433 6.133337696563011
0.026393761040726486 6.220269270650224
0.0026451135254408876 6.228990096672928
0.00026456875298537204 6.229862455971611
```

[21]:
```
#4c) 45.819
riemann(d, 3, 7, 1000000, "left")
```

```
7.418888455484781e-05 45.81998520340121
```

[22]:
```
#4d) 52.049
#at 1000000 intervals, the limiting values stabilize out to 3 decimal points
#intervals all have at least 3 zeros after the decimal
riemann(d, 1, 7, 1000000, "left")
```

```
0.00011128327928204311 52.04991970421313
```

5) Added another parameter to be able to choose left, mid, or right.

for midpoint, we have to use x = a + dx / 2 instead of just x = a.

[23]:
```
#6a) by 40000 intervals, 7 decimal points have stabilized
riemann(d, 1, 3, 40, "mid")
riemann(d, 1, 3, 400, "mid")
riemann(d, 1, 3, 4000, "mid")
riemann(d, 1, 3, 40000, "mid")
```

```
0.2613934265384266 6.229804122734282
0.02642562971547407 5 6.229957835175769
0.002645432410896677 6.229959372356611
0.00026457194203854576 6.229959387732263
```

6b) 40 intervals stabilizes to first 4 digits

6.229

the two values should be close to the same, given higher numbers of intervals

6c) midpoint Riemann sums seems much more efficient than left endpoint (and I would guess more efficient than the right endpoint as well, but I haven't really tested that hypothesis, yet)

[24]:
```
#7a) seems as though 3, maybe 2-3 digits have stabilized.
riemann(d, 1, 3, 40, "right")
```

```
riemann(d, 1, 3, 400, "right")
riemann(d, 1, 3, 4000, "right")
riemann(d, 1, 3, 40000, "right")
```

```
0.26457513110645875 6.327202149550815
0.026457513110645363 6.239655715949005
0.002645751311064876 6.230928741202806
0.00026457513110643096 6.2300563204246
```

7b) right endpoint calculations seem to be the least efficient of the 3

[25]:
```
#8) my values match the ones on page 356
e = lambda x: math.sqrt(1-x**2)
riemann(e, -1, 1, 20, "left")
riemann(e, -1, 1, 50, "left")
```

```
0.04358898943540679 1.5522591631241593
0.011199999999999915 1.5660981554514977
```

[26]:
```
#9a) my values match the ones on page 358
i = lambda x: math.sqrt(1+math.cos(x)**2)
riemann(i, 0, math.pi, 4, "left")
riemann(i, 0, math.pi, 20, "left")
```

```
0.961912372621398 3.819943643179836
0.22078090039159703 3.820197789027713
```

[27]:
```
#9b) 3.82019778902
riemann(i, 0, math.pi, 1000000, "left")
```

```
4.442882938147403e-06 3.8201977890251326
```

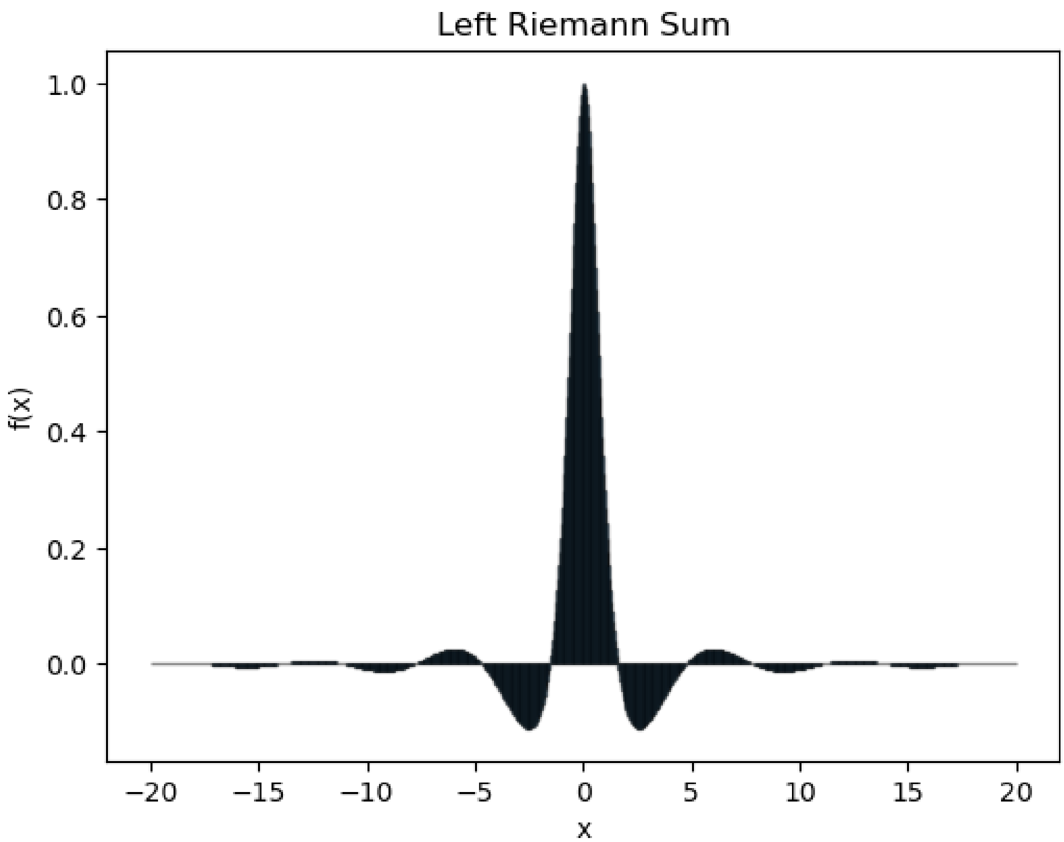
[28]:
```
#10
p = lambda x: math.cos(x**2)
riemann(p, 0, 4, 100, "left")
riemann(p, 0, 4, 1000, "left")
riemann(p, 0, 4, 10000, "left")
```

```
-0.039986100367134784 0.6339209745734815
-0.0038655056816245016 0.5983787174632444
-0.00038343032823155644 0.5948518901039331
```

[29]:
```
#11) cool graph!
q = lambda x: math.cos(x) / (1+x**2)
riemann(q, 2, 3, 10, "left")
riemann(q, 2, 3, 100, "left")
riemann(q, 2, 3, 1000, "left")
```

```
#curiosity got the best of me, so I changed the interval to see the symmetry␣
↪around 0
riemann(q, -20, 20, 1000, "left", plot=True)
```

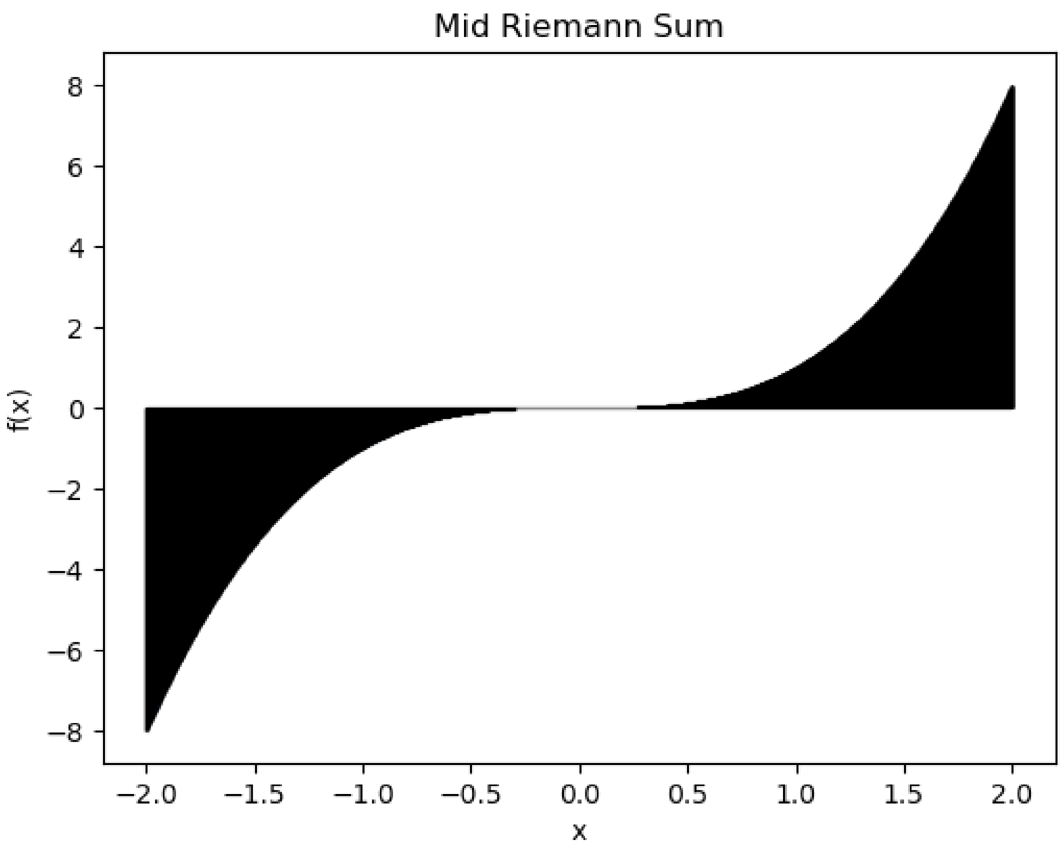
```
-0.010318365198189058 -0.10320774749944361
-0.00099448880879207 -0.10404981921880206
-9.904450496050063e-05 -0.10412210833479739
4.449293266890143e-05 1.1600178434599466
```




```
[30]: #12a) the sums are zero because we have equal amounts on both negative and␣
↪positive sides of zero
#multiplying 3 negative numbers gives a negative product, so if x is negative,␣
↪y is negative
#multiplying 3 positive numbers gives a positive product, so if x is positive,␣
↪y is positive
h = lambda z: z**3
riemann(h, -2, 2, 10, "mid")
riemann(h, -2, 2, 100, "mid")
riemann(h, -2, 2, 1000, "mid")
#curiosity again. I'm a visual learner; I just like seeing the plots
```

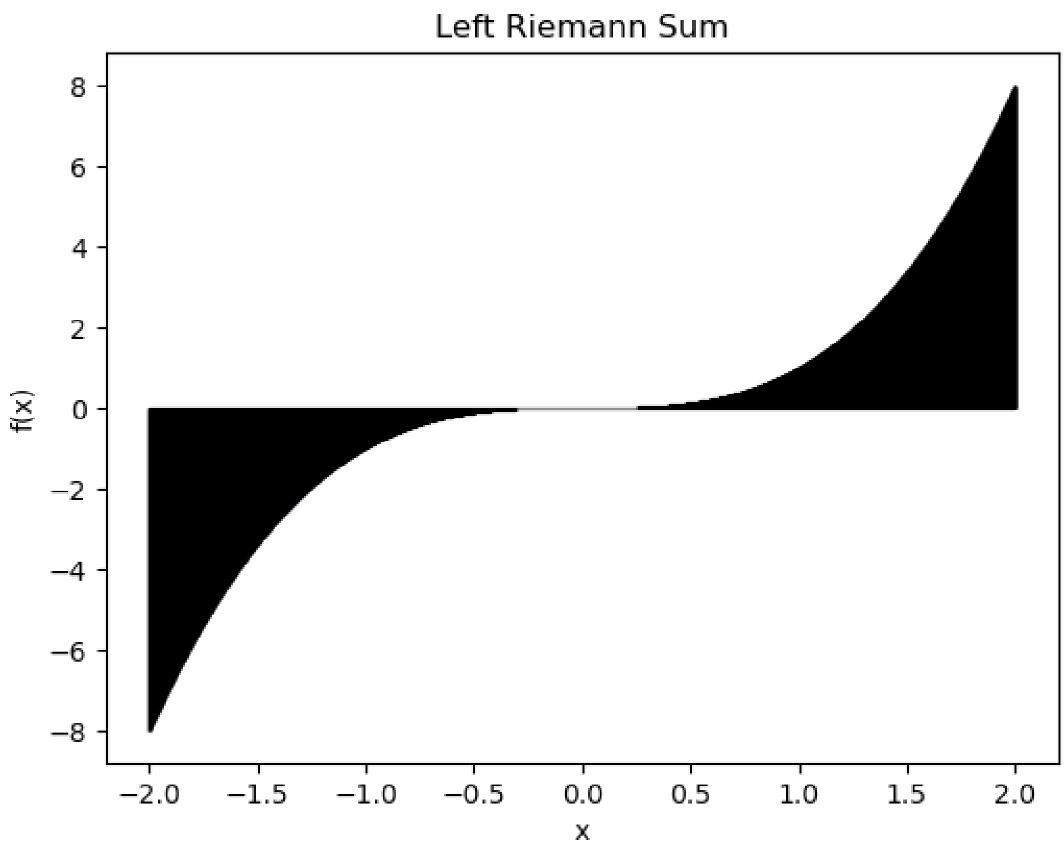
```
riemann(h, -2, 2, 10000, "mid", plot=True)
```

```
2.332800000000001 3.1086244689504383e-15
0.31049568000000116 2.020605904817785e-14
0.03190409596800016 2.896988204881268e-14
0.0031990400959953126 -2.4963334351035815e-12
```



[31]:
```
#12b) the results are not still zero, but the graph still shows the same␣
↪symmetry,
#so the sums SHOULD be zero, right?
riemann(h, -2, 2, 10, "left")
riemann(h, -2, 2, 100, "left")
riemann(h, -2, 2, 1000, "left")
#curiosity again. I'm a visual learner; I just like seeing the plots
riemann(h, -2, 2, 10000, "left", plot=True)
```

```
1.6384000000000005 -3.200000000000001
0.30118144000000113 -0.3199999999999822
0.03180838374400016 -0.03199999999997637
0.003198080383972913 -0.0032000000024886167
```



[32]:
```
# Activity 4

def simpson(func, start, stop, steps):
    f = func
    a = start
    b = stop
    dx = (b - a) / steps

    acc_left = 0
    acc_right = 0
    acc_mid = 0

    #calculating left sum
    x_left = a
    for k in range(steps):
        ds_left = f(x_left) * dx
        acc_left += ds_left
        x_left += dx

    #calculating right sum
```

```
    x_right = a + dx
    for k in range(steps):
        ds_right = f(x_right) * dx
        acc_right += ds_right
        x_right += dx

    #calculating midpoint sum
    x_mid = a + dx / 2
    for k in range(steps):
        ds_mid = f(x_mid) * dx
        acc_mid += ds_mid
        x_mid += dx

    #calculating Simpson's rule
    simp = (4*acc_mid + acc_left + acc_right) / 6

    return simp
```

[33]:
```
simpson(g, 3, 10, 30)
```

[33]: 120.51610120419456

[34]:
```
riemann(g, 3, 10, 30, "mid")
```

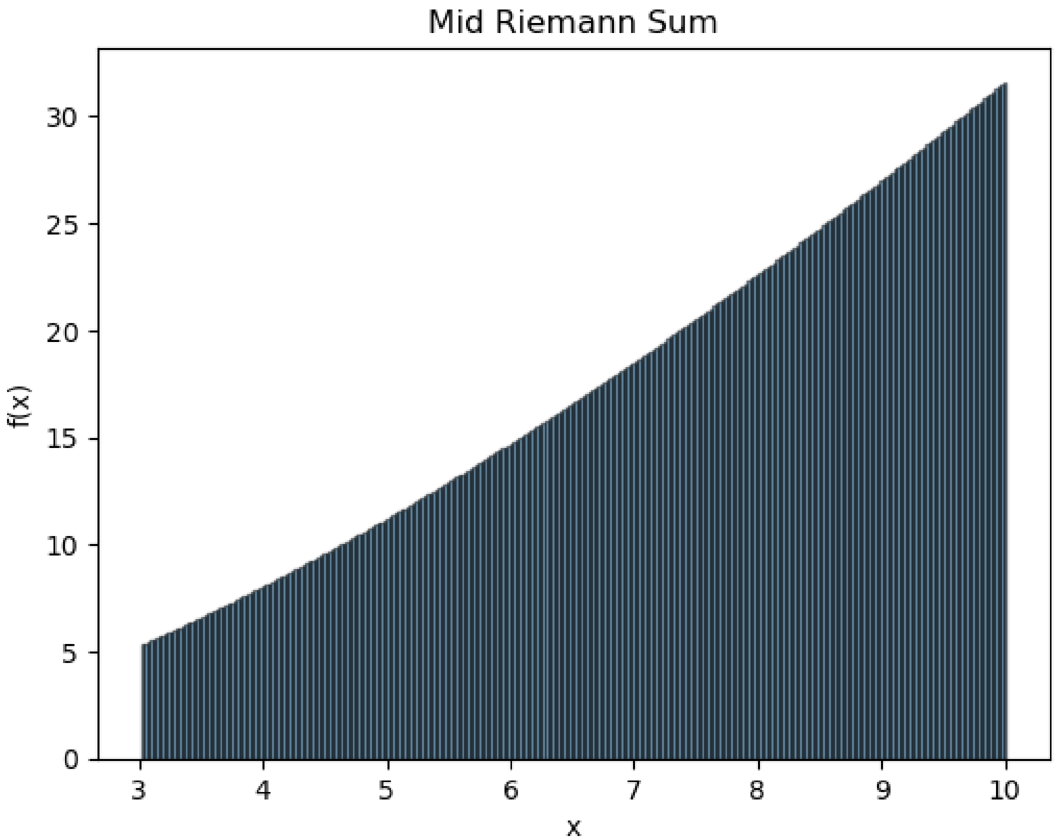
7.253652759751697 120.5111338950847

[37]:
```
#here I'm just messing around with the function,
#seeing if there's any major difference if I only update
#heights and x_vals in the loops if plot=True

def rie(func, start, stop, steps, point, plot=True):
    f = func
    a = start
    b = stop
    dx = (b - a) / steps
    acc = 0
    heights = []
    x_vals = []

    if point == "left":
        x = a
        for k in range(steps):
            ds = f(x) * dx
            acc += ds
            if plot:
                heights.append(f(x))
                x_vals.append(x)
```

```
            x += dx
        print(ds, acc)

    elif point == "mid":
        x = a + dx / 2
        for k in range(steps):
            ds = f(x) * dx
            acc += ds
            if plot:
                heights.append(f(x))
                x_vals.append(x)
            x += dx
        print(ds, acc)

    elif point == "right":
        x = a + dx
        for k in range(steps):
            ds = f(x) * dx
            acc += ds
            if plot:
                heights.append(f(x))
                x_vals.append(x)
            x += dx
        print(ds, acc)

    if plot:
        plt.bar(x_vals, heights, width=dx, align='edge', edgecolor='black',␣
↪alpha=0.5)
        plt.xlabel('x')
        plt.ylabel('f(x)')
        plt.title(f'{point.capitalize()} Riemann Sum')
        plt.show()
```

[38]:
```
rie(g, 3, 10, 300, "mid")
```

```
0.7369433869822895 120.51605142383
```


Mid Riemann Sum
f(x)
x
0
5
10
15
20
25
30
3
4
5
6
7
8
9
10

```
[ ]:
```