# Adaptive Integration

A Numerical Methods Lecture Using Python by Mark E. Redd

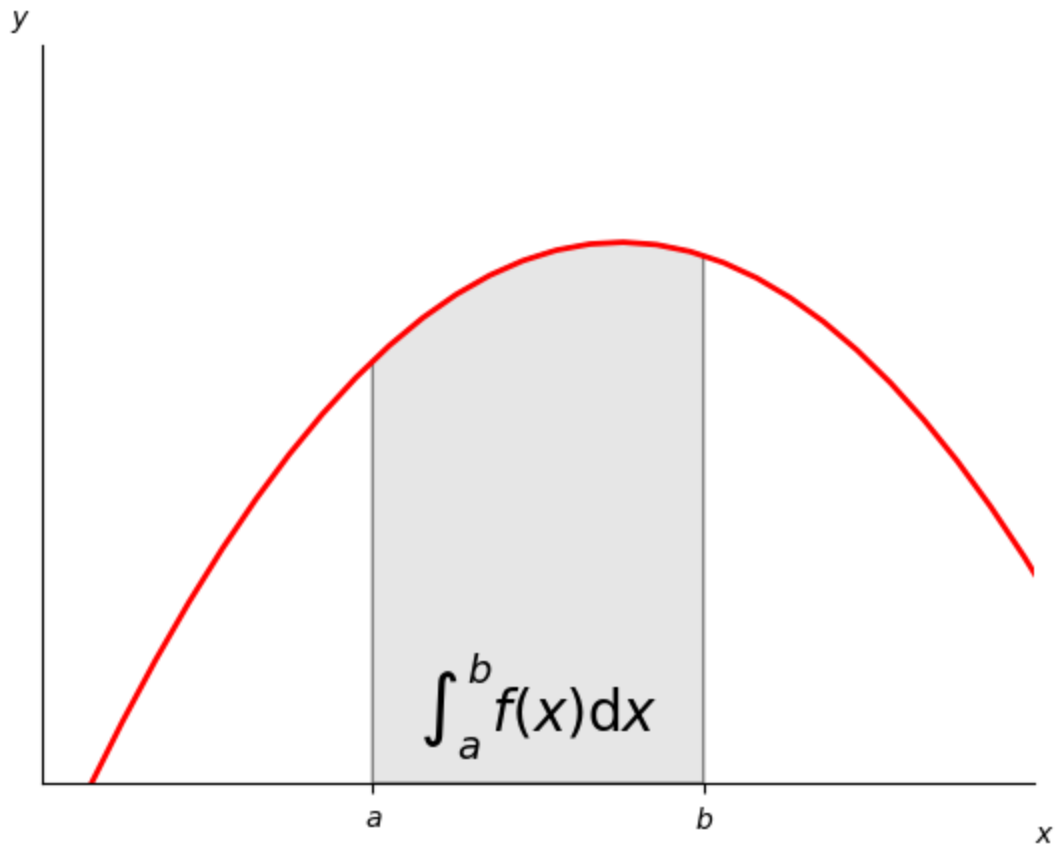
# Introduction

Suppose we wish to integrate a function. That is, we wish to find the area under a certain section of a curve.
That might look something like this:

$$\int_a^b f(x)\mathrm{d}x$$

or more specifically, where $f(x) = -2x^2 + 3x + 4$ and $a, b = 0, 1$:

$$\int_0^1 -2x^2 + 3x + 4 \, \mathrm{d}x$$


y
$\int_a^b f(x)\mathrm{d}x$
a
b
x

Now, this is straight forward to do if we understand the basic rules of polynomial integreation. The solution to this integral is:

$$\frac{dF(x)}{dx} = f(x)$$

$$F(x) = \frac{-2}{3}x^3 + \frac{3}{2}x^2 + 4x$$

$$\int_a^b f(x)\mathrm{d}x = F(b) - F(a)$$

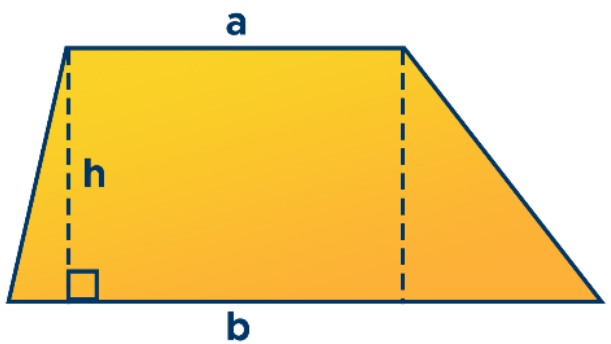
$$\int_0^1 -2x^2 + 3x + 4\,\mathrm{d}x = \frac{-2}{3}(1)^3 + \frac{3}{2}(1)^2 + 4(1) - \left(\frac{-2}{3}(0)^3 + \frac{3}{2}(0)^2 + 4(0)\right) = \frac{-2}{3} + \frac{3}{2} + 4 - 0 = 4.8\overline{333}$$

But what if we want a computer to do this for us? What if we want to integrate any arbitrary function?

## The Trapezoid Rule

Well, you may already be familiar with the trapazoid rule. Simply split the area to be integrated into trapazoids and add up the area of the trapazoids. Recall that the area of a trapazoid is:

# Area of a Trapezoid




$$A = \frac{a + b}{2} h$$

**Where:**

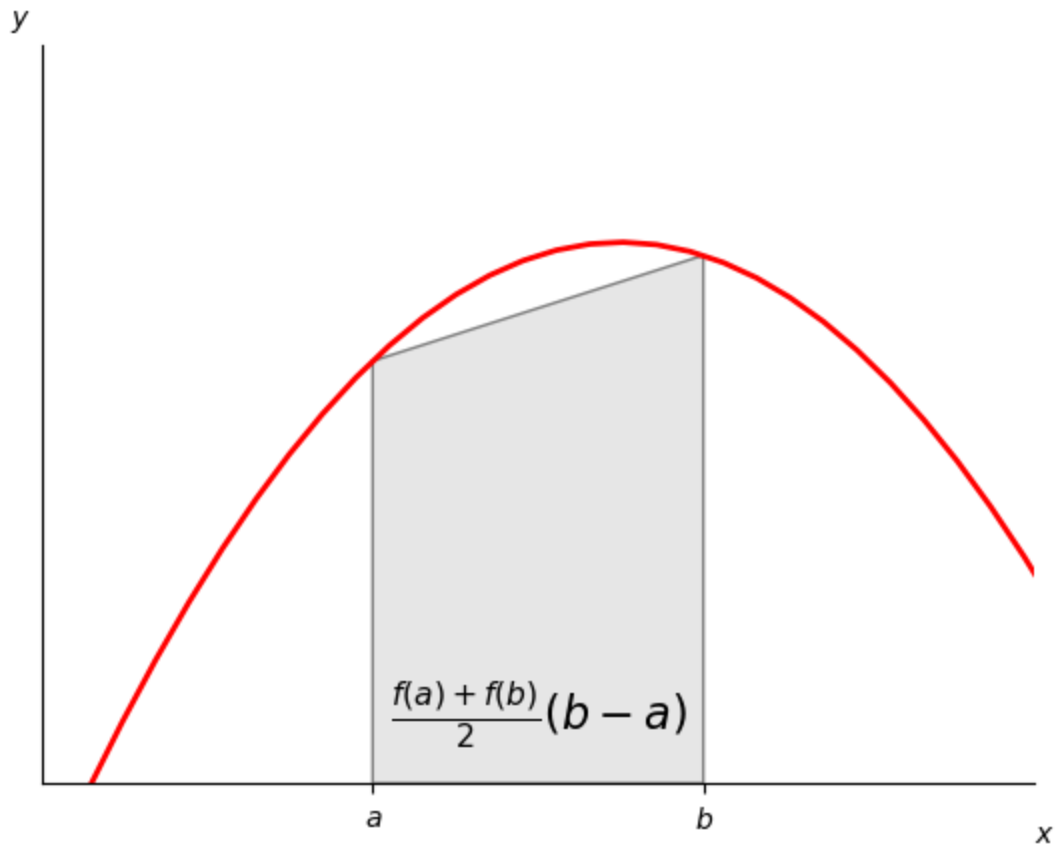
***A*** **= Area of a Trapezoid**

***a*** **= length of base 1**

***b*** **= length of base 2**

***h*** **= height of the trapezoid**

And if we take the two base lengths $a$ and $b$ of the trapazoid to be $f(a)$ and $f(b)$ and $h$ to be $(b-a)$ or the distance between the two points we get something that looks like this:

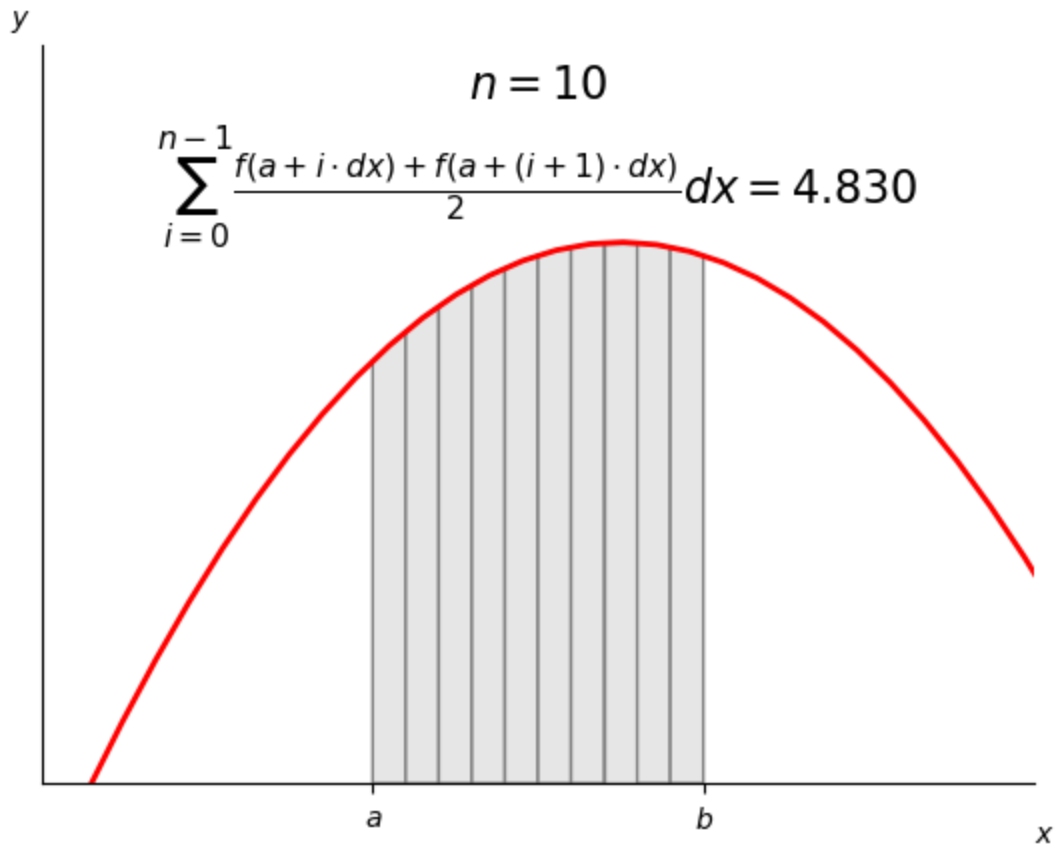

y
$\frac{f(a)+f(b)}{2}(b-a)$
a
b
x

This gives a value of $4.75$ which is a pretty good first approximation of the actual integral ($\approx 17\%$ relative error). If we want to, we can just break the trapazoids into pieces and get a more accurate answer. That is:

$$dx = \frac{b-a}{n}$$

$$\sum_{i=0}^{n-1} \frac{f(a+i\cdot dx) + f(a+(i+1)\cdot dx)}{2} dx$$

This makes our trapazoid look like this:


y
$n = 10$
$\sum_{i=0}^{n-1} \frac{f(a + i \cdot dx) + f(a + (i+1) \cdot dx)}{2} dx = 4.830$
a
b
x

Using $n = 10$ number of trapazoids gives $4.830$, a much better approximation. We can increase $n$ until we get to a desired accuracy.

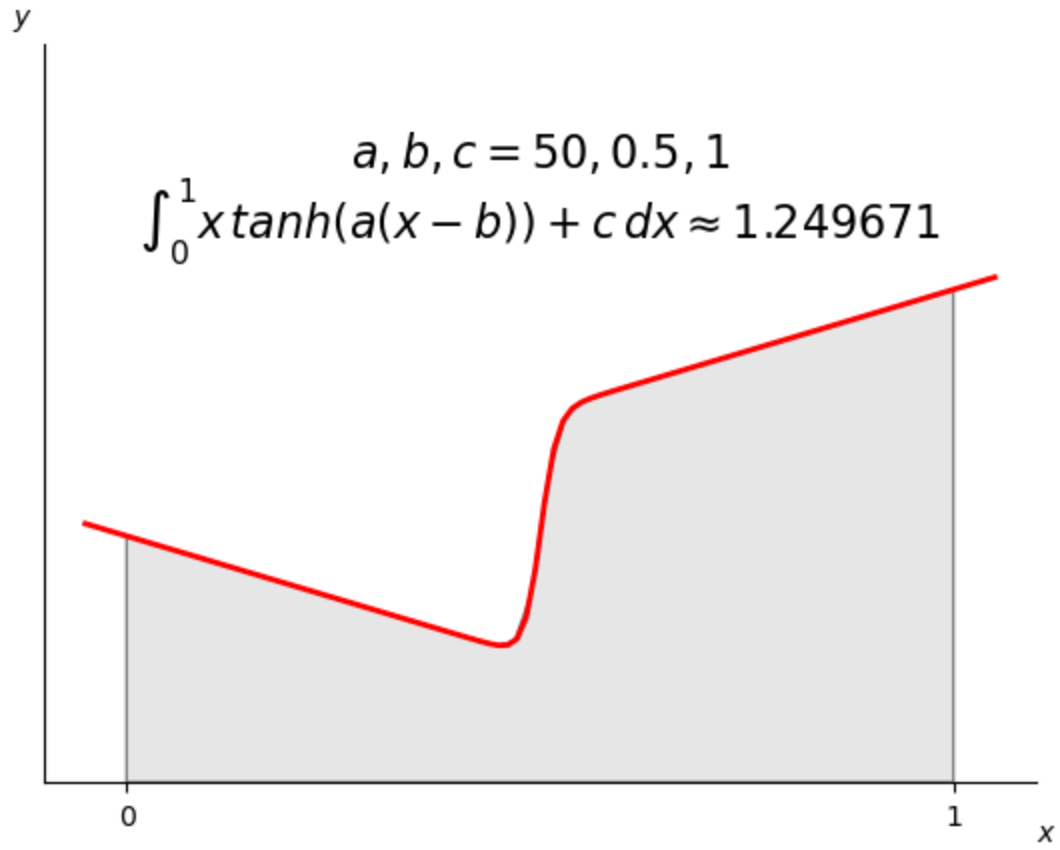
```
for i in range(5):
    n=10**(i+1)
    print(f"trapezoid rule (n={n:7d}): {trapezoid_rule(easy_func, 0, 1, n=n):.10f}")
```

```
trapezoid rule (n=     10): 4.8300000000
trapezoid rule (n=    100): 4.8333000000
trapezoid rule (n=   1000): 4.8333330000
trapezoid rule (n=  10000): 4.8333333300
trapezoid rule (n= 100000): 4.8333333333
```

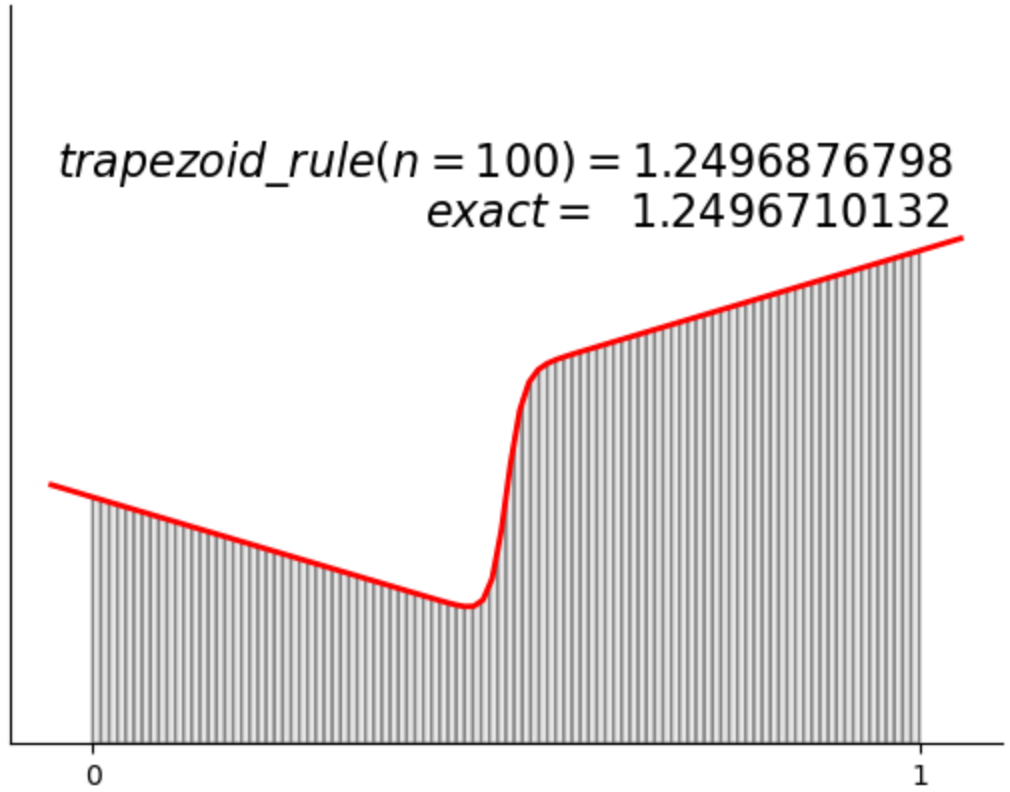
We see that at $n = 100{,}000$ we get 10 decimals of accuracy. This is good, but can be taxing on a computer. Especially when you need to do many accurate integrations for some complex calculation. There is another problem too. What if we have an integral that looks like this?

$$\int_0^1 x\, tanh\left(a\left(x - b\right)\right) + c\, dx$$

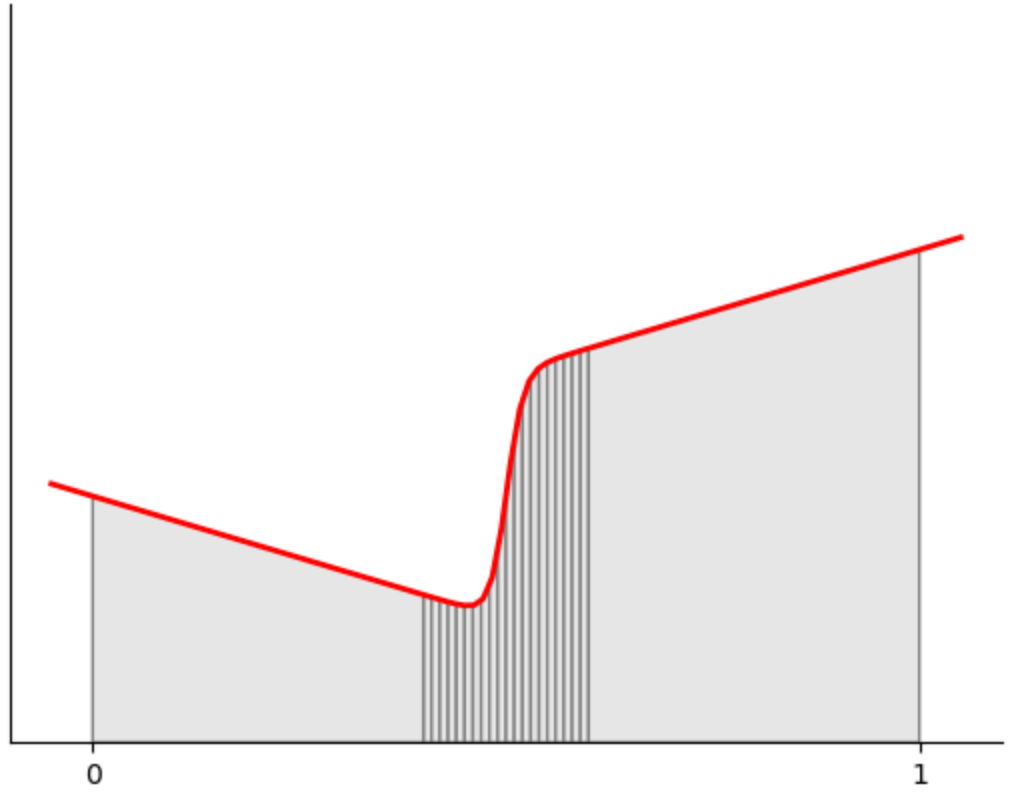
This one is not easy to do by hand. It also looks like this on a graph:



Also to integrate using the trapazoid rule can be a problem. Splitting the integral into 10,000 trapezoids gets an accurate answer but is slow. It is a complex fuction that is computationally expensive to calcualte.

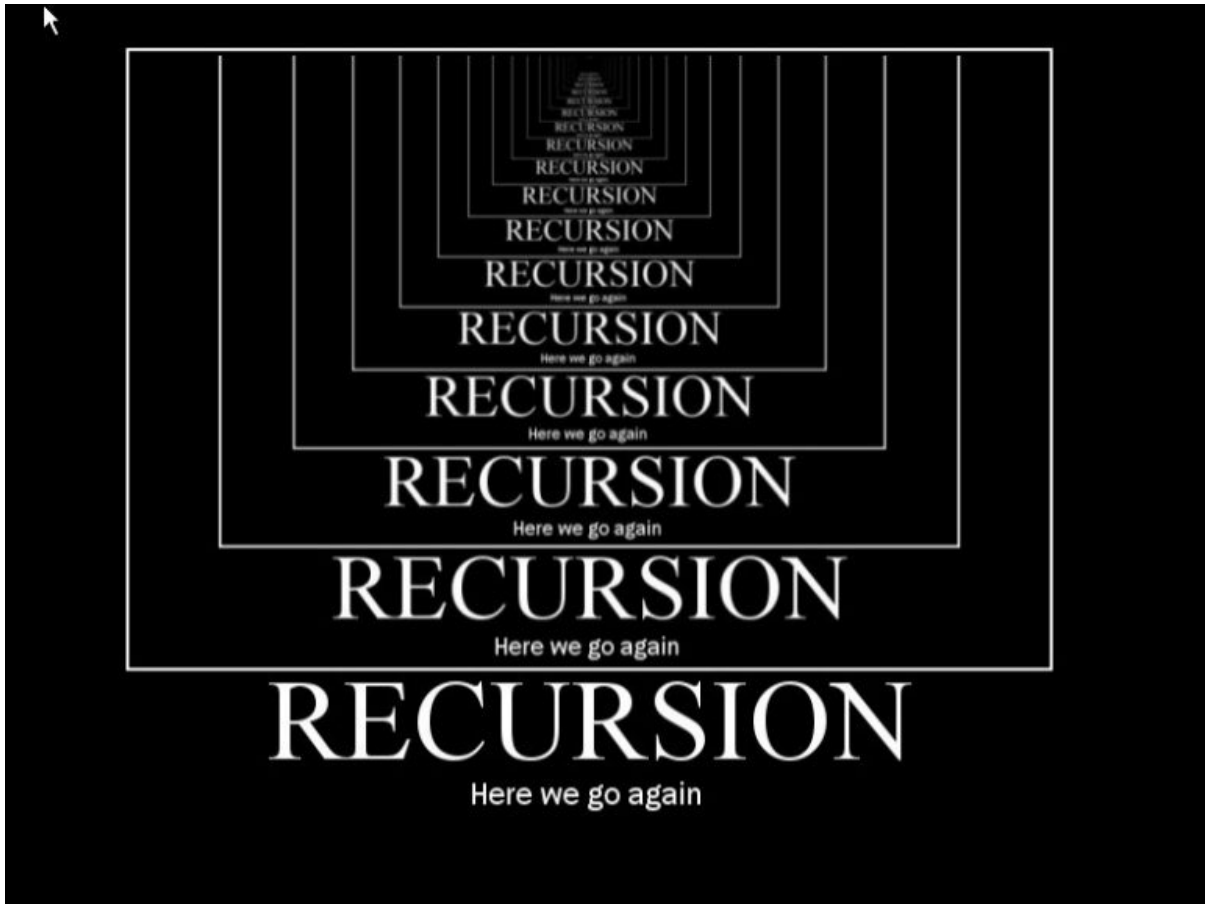


But, consider this: What if we could choose how big we wanted each trapezoid could be? If it were up to me I would do it like this:




You see, this way we could do this in many fewer iterations and get a very accurate answer! But how do we program a computer to figure this out?

# Recursion!



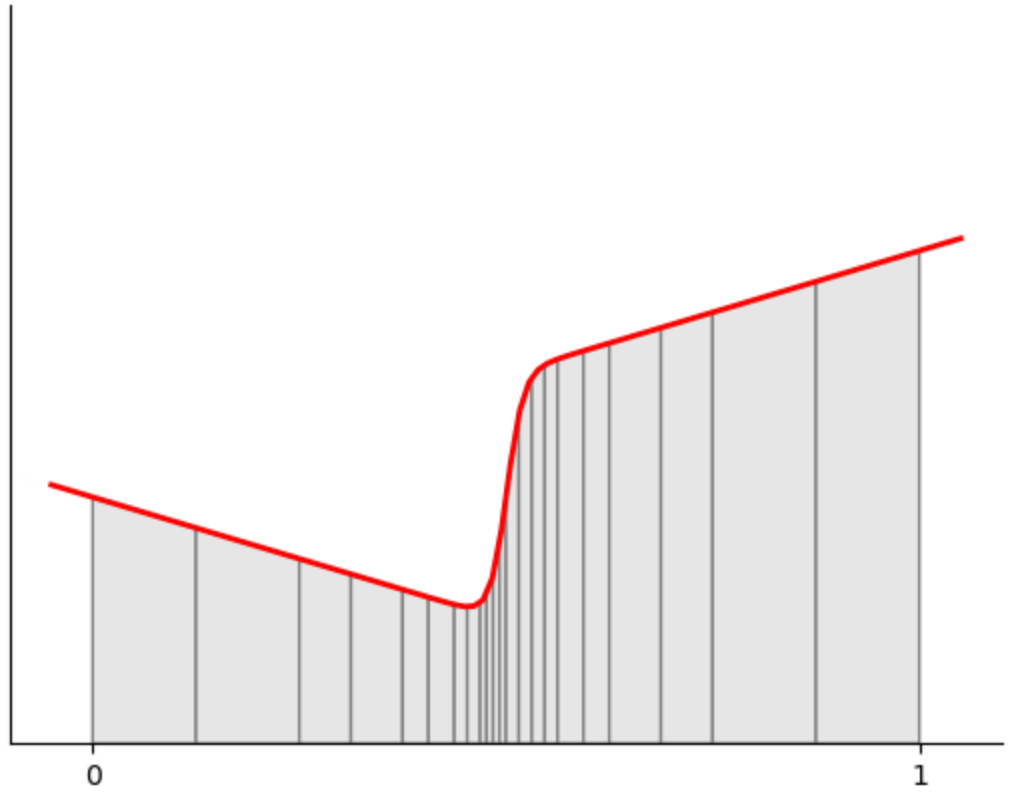

We can write a recursive function that will automatically and correctly choose the correct spacing for difficult functions like the one above! Lets try it!

```
# code it here!
# def trap_adapt():
#     pass
```

```
print(f"trap:       {trapezoid_rule(easy_func, 0, 1, n=10):.10f}")
print(f"trap_adapt: {trap_adapt(easy_func, 0, 1):.10f}")
print(f"Scipy:      {sp.integrate.quad(easy_func, 0, 1)[0]:.10f}")
```

```
trap:       4.8300000000
trap_adapt: 4.8333282471
Scipy:      4.8333333333
```

```
Integral value: 1.24936 (0.02505% error); 22 trapezoids
```



```
print(f"trap:       {trapezoid_rule(hard_func, 0, 1, n=10000):.10f}")
print(f"trap_adapt: {trap_adapt(hard_func, 0, 1):.10f}")
print(f"Scipy:      {sp.integrate.quad(hard_func, 0, 1)[0]:.10f}")
```

```
trap:       1.2496710149
trap_adapt: 1.2496706792
Scipy:      1.2496710132
```

## Push this further (1 of 2)

Try the following to deepen your knowledge:

- We have only used one example function for this algorithm. Try using the adaptive trapazoid method to integrate various functions. Are there any potential problems with this method? Try to find an example where it may give you a wrong calculated integral.
- Try rewriting the adaptive algorithm using Simpson's Rule. In what ways is it better? faster? Compare it to the trapazoid version and explain any advantages or drawbacks to using your rewritten function.

## Push this further (2 of 2)

Try the following to deepen your knowledge:

- Usually, recursive programming is not ideal for stablity of programs. Could you write the adaptive trapazoid method without recursion? What are the advantages and drawbacks to this?
- Research how these algorithms are used in production code. Look up adaptive quadrature or research the Python function scipy.integrate.quad. How are these methods different to the ones we programmed here? Are there any drawbacks to their methods?