Adaptive Integration

A Numerical Methods Lecture Using Python by Mark E. Redd

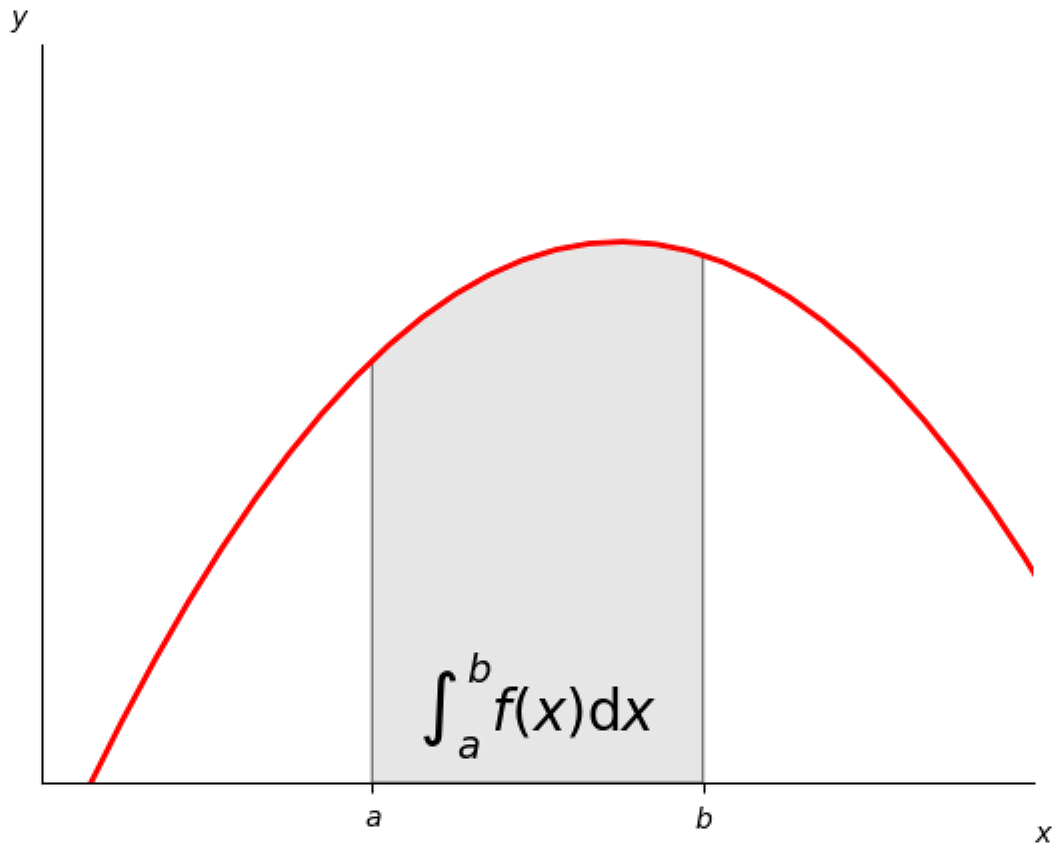
Introduction

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

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

$$f(x) = -2x^2 + 3x + 4;\ a, b = 0, 1$$

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



The solution to this integral is:

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

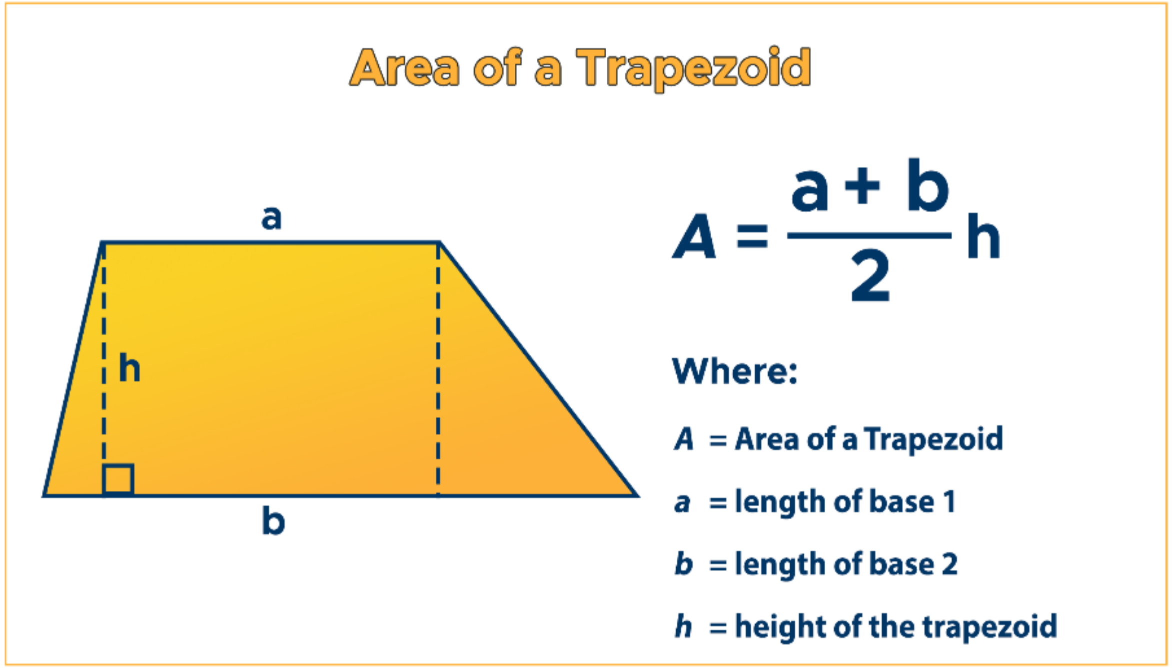
$$F(x) = \frac{-2}{3}x^3 + \frac{3}{2}x^2 + 4x$$

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

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

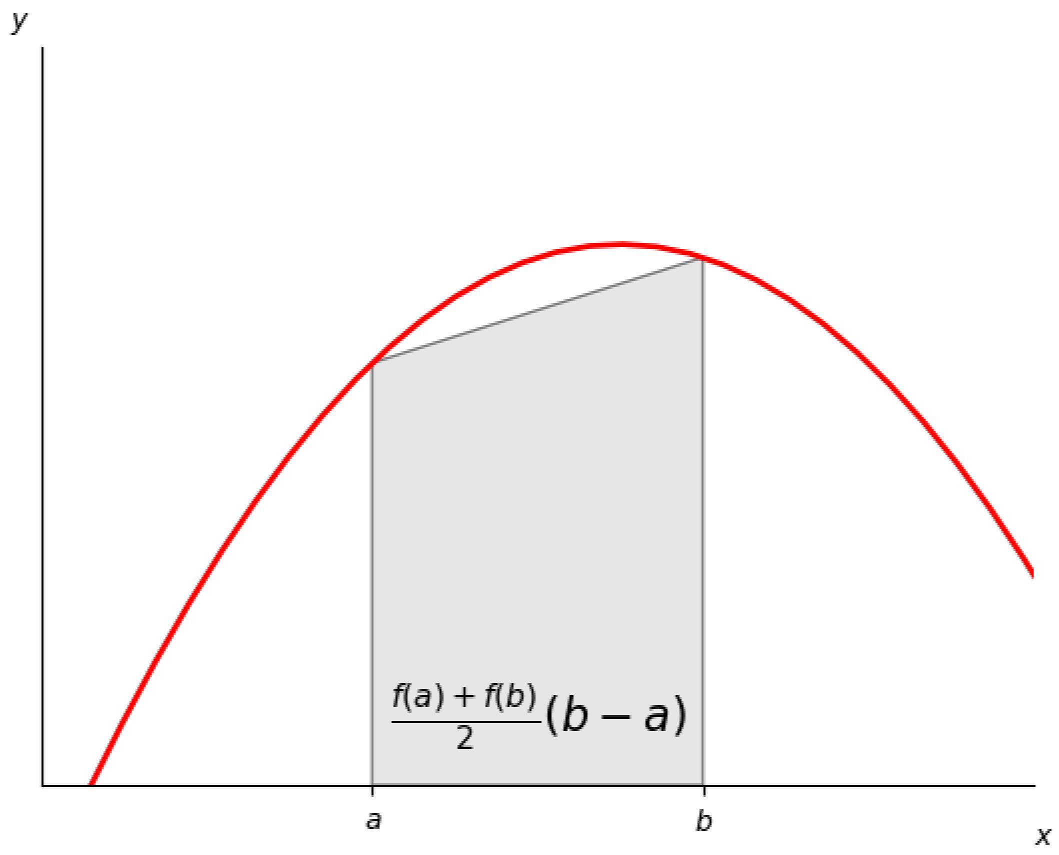
But what if we want a computer to do this for us?

The Trapezoid Rule



Exact is $4.8\overline{333}$

$$\frac{f(a)+f(b)}{2}(b-a) = 4.75 \; (\approx 17\% \; relative \; error)$$

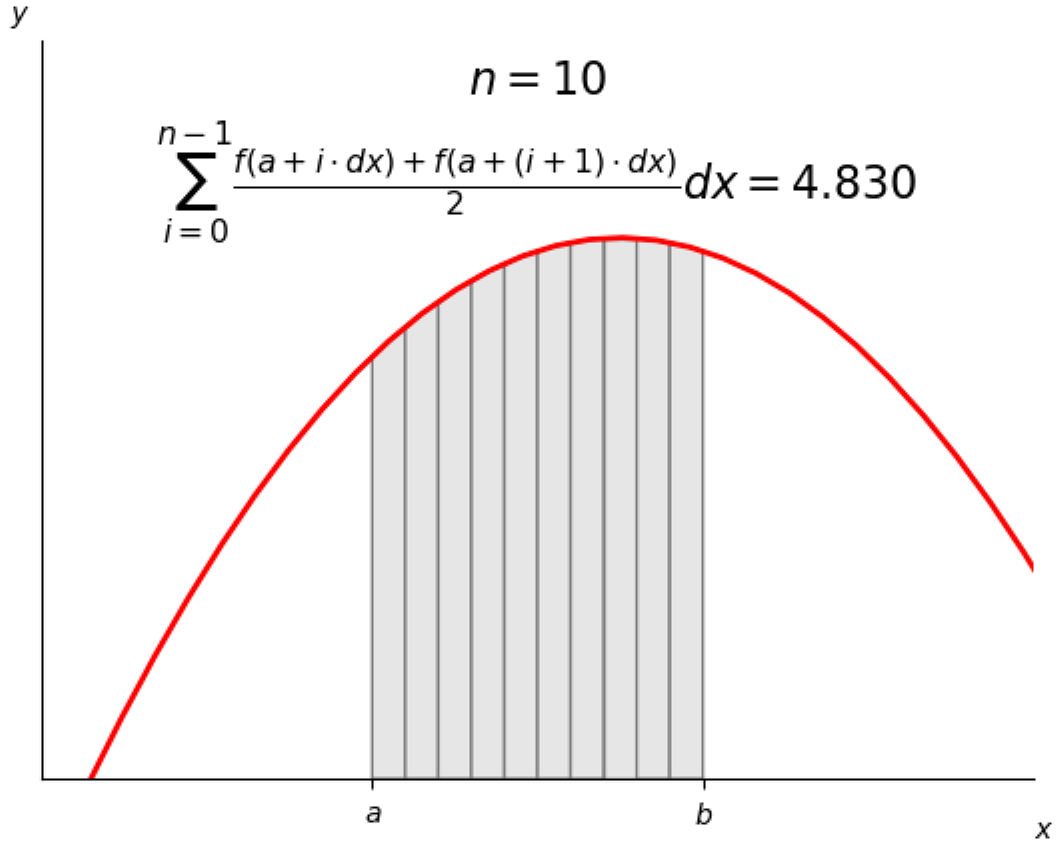


If we want to, we can just use more, smaller trapazoids and get a more accurate answer.

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

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

This makes our trapazoid look like this:



- $n = 10\ trapazoids \rightarrow 4.830$, 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}")
```

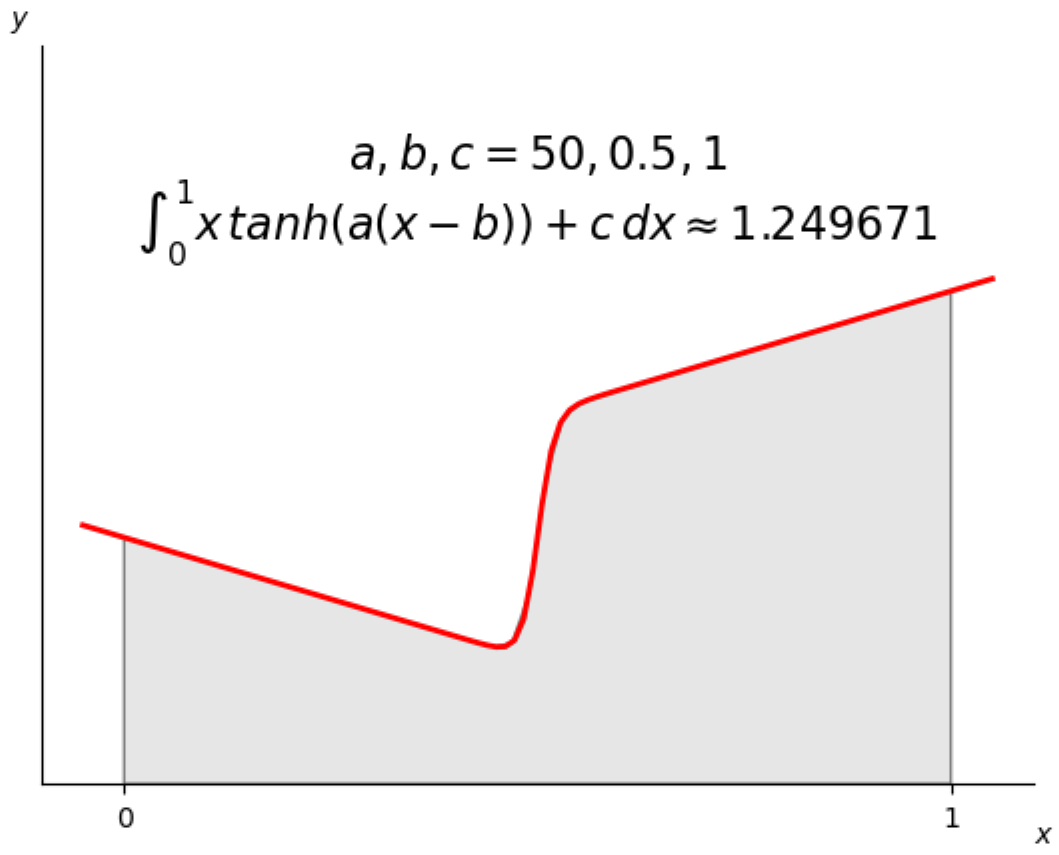
```
trapezoid rule (n=     10): 4.8300000000
trapezoid rule (n=    100): 4.8333000000
trapezoid rule (n=   1000): 4.8333330000
trapezoid rule (n=  10000): 4.8333333300
trapezoid rule (n= 100000): 4.8333333333
```

- $n = 100{,}000$ we get 10 decimals of accuracy.
- This is good, but can be taxing on a computer.

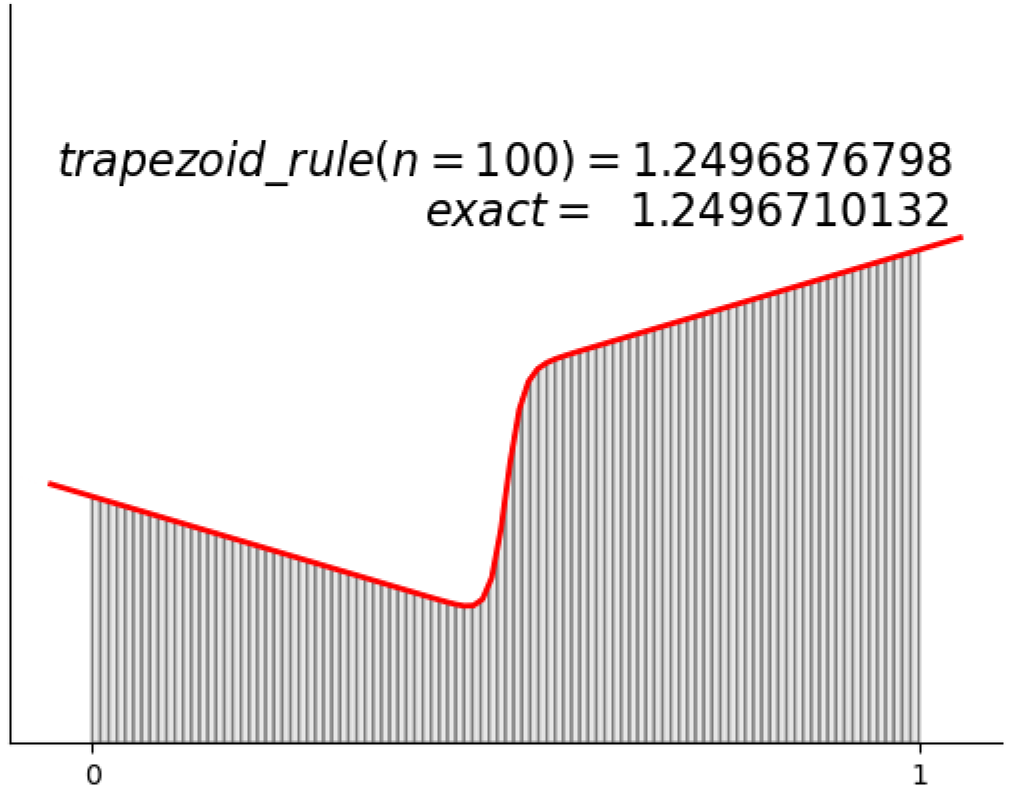
But, what if we have an integral that looks like this?

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

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

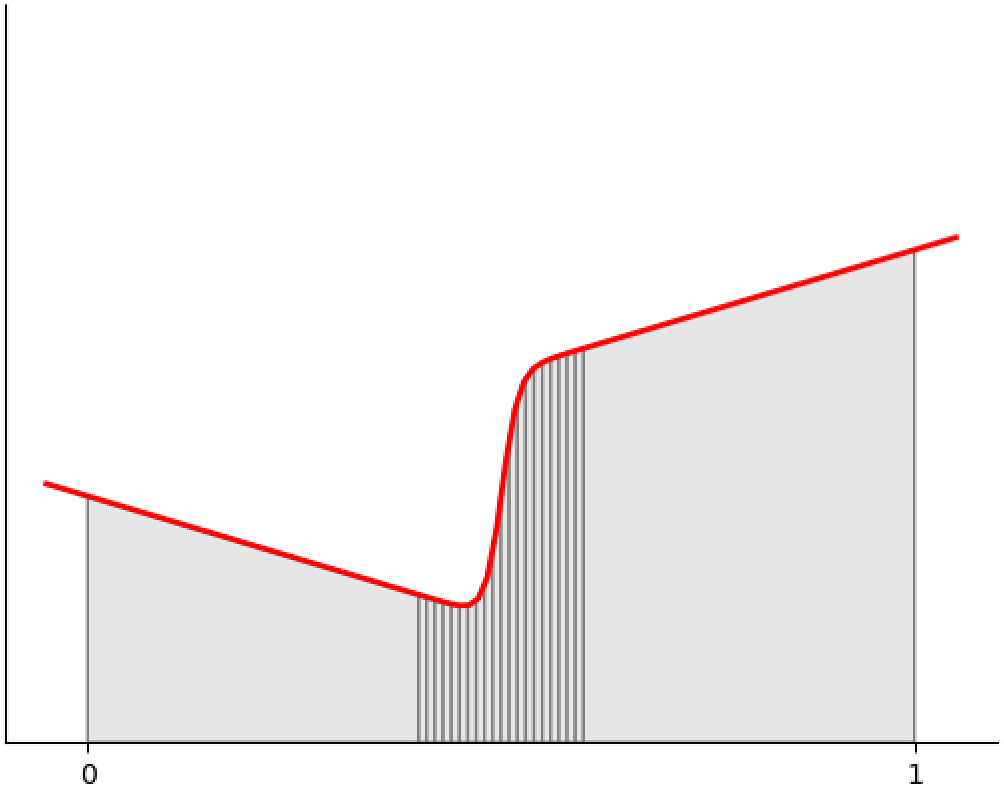


- $n = 10{,}000$ trapezoids gets an accurate answer but is slow
- Complex fuction, computationally expensive to calcualte

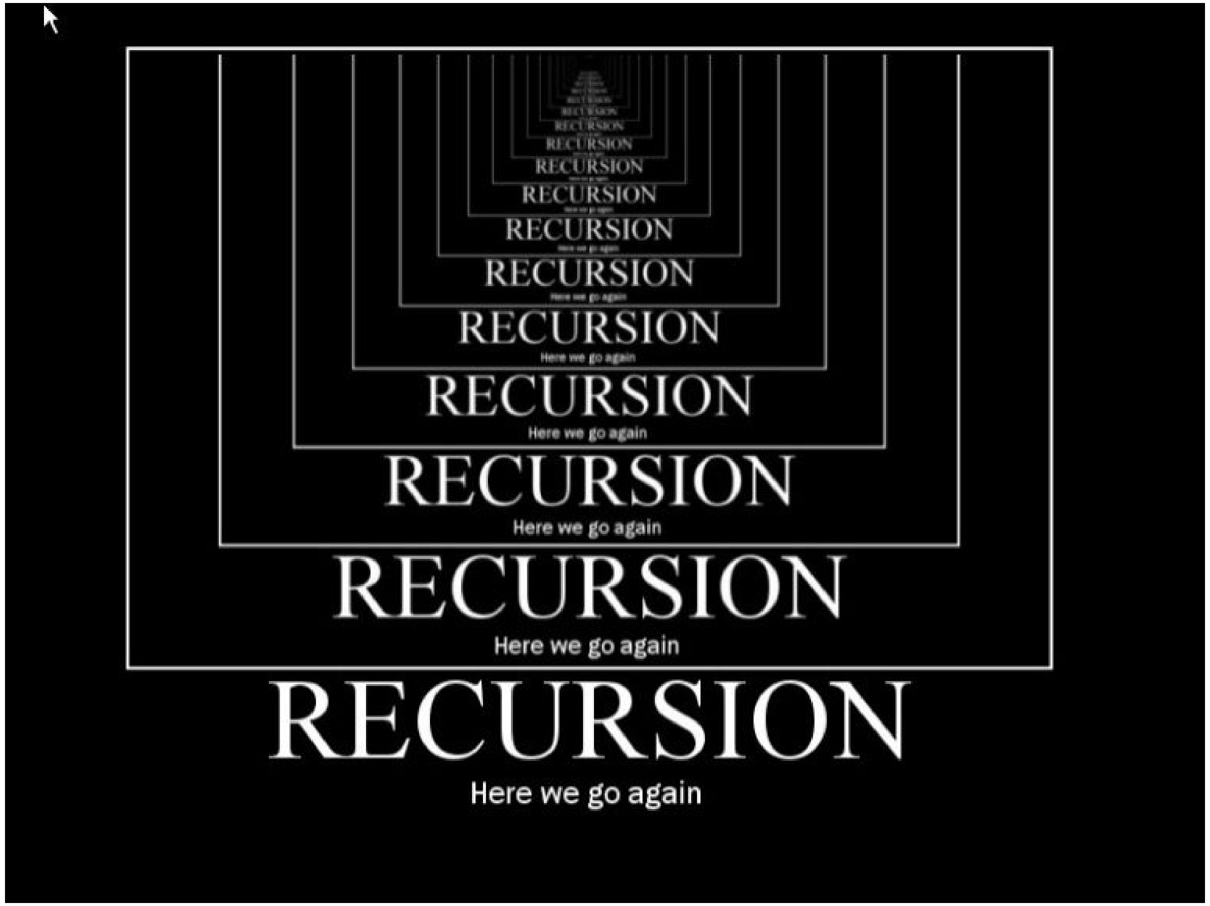


Consider this:

- What if we could choose how big we wanted each trapezoid could be?
- I would do it like this:



Recursion!

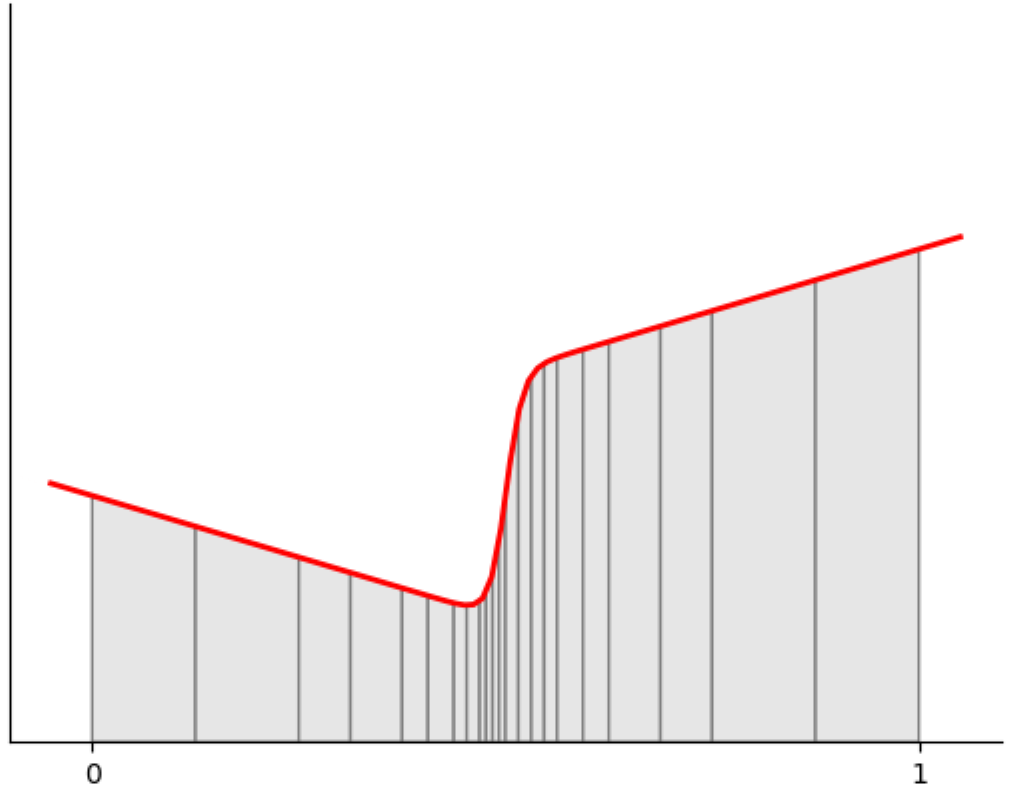

RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
RECURSION
RECURSION
RECURSION
RECURSION

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

```python
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
```



```python
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?