Numerical Integration PHYS4840

Trapz, Simpsons, Romberg, errors

github.com/nialljmiller/PHYS4840_Numerical_integration_I
(This is currently the pinned repo on my page)

Dr Niall Miller

Office hours - 2 hours before class or by request

Numerical Integration

- Why do we need numerical integration methods?
- Why is there more than one?
- Why is my boss mad at me for using trapz?
- Cant AI do this?

"There is no known way to calculate the area under a curve exactly in all cases on a computer" - Mark Newman

Numerical Integration

- Why do we need numerical integration methods?
- Why is there more than one?
- Why is my boss mad at me for using trapz?
- Cant AI do this?

“There is no known way to calculate the area under a curve exactly in all cases on a computer” - Mark Newman

Computers will never integrate

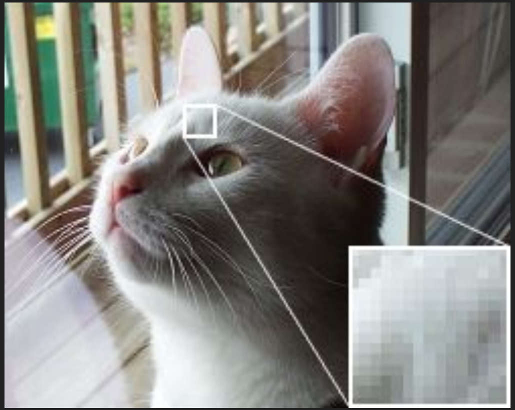
- Computers may integrate into society one day
- However, as computers can only ever speak in a discretized way, they won't ever be able to compute pure integration - or anything in a truly continuous way
- Everything a computer does to interface with the real world is based on this

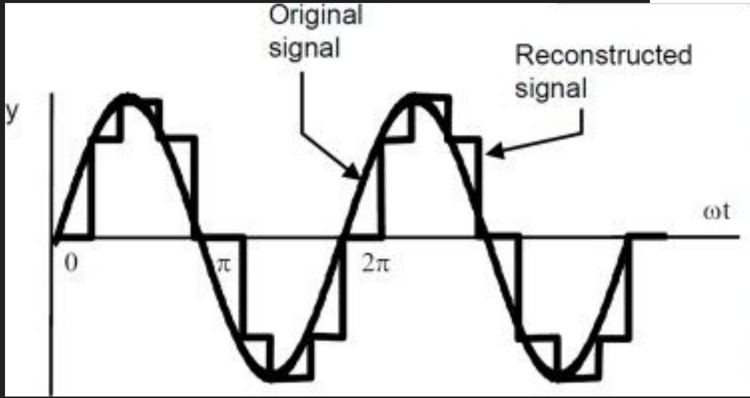
Computers will never integrate

- This is a fundamental limitation of computing

All the visuals we see rendered by a computer are quantized into pixels.

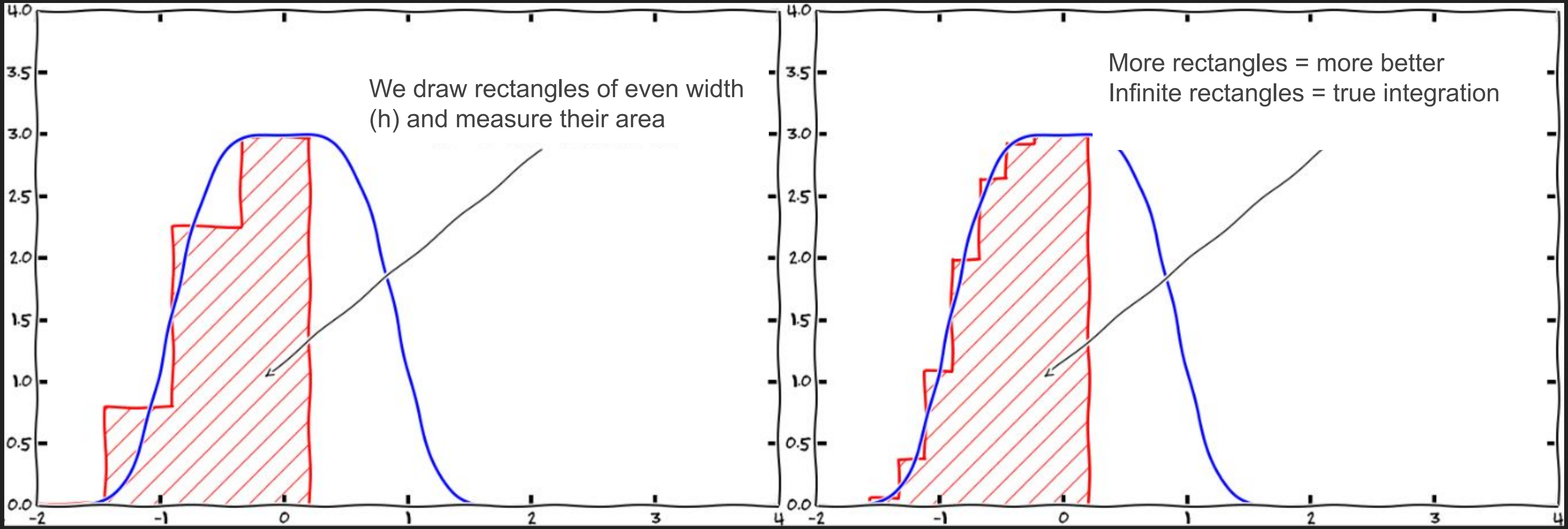
Any audio we hear from a computer must first go through a Digital to Analogue Converter.





So we approximate integration

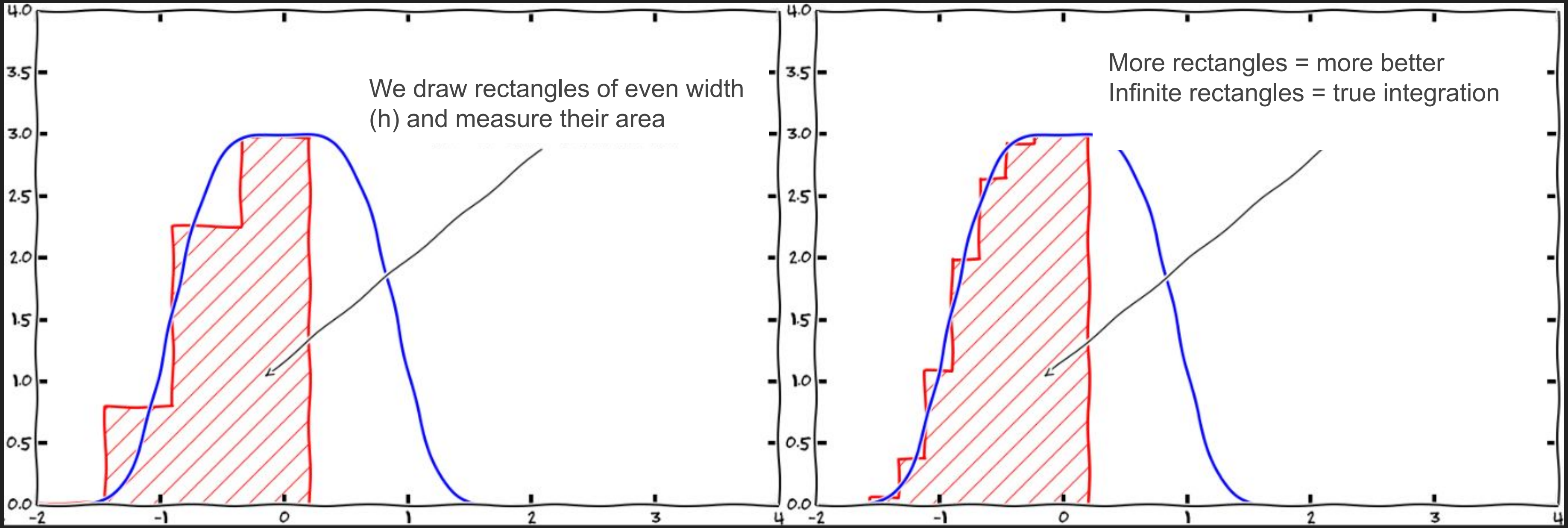
- Computers may never be able to comprehend the beauty of the analogue world
- But they are really really fast and good at discrete maths.



https://link.springer.com/chapter/10.1007/978-3-030-16877-3_6

So we approximate integration

- So what if we want to find the area under a continuously defined curve (f(x))?



https://link.springer.com/chapter/10.1007/978-3-030-16877-3_6

Trapezoidal rule

- Slight improvement to the rectangles (this is foreshadowing)
- Now for some Maths

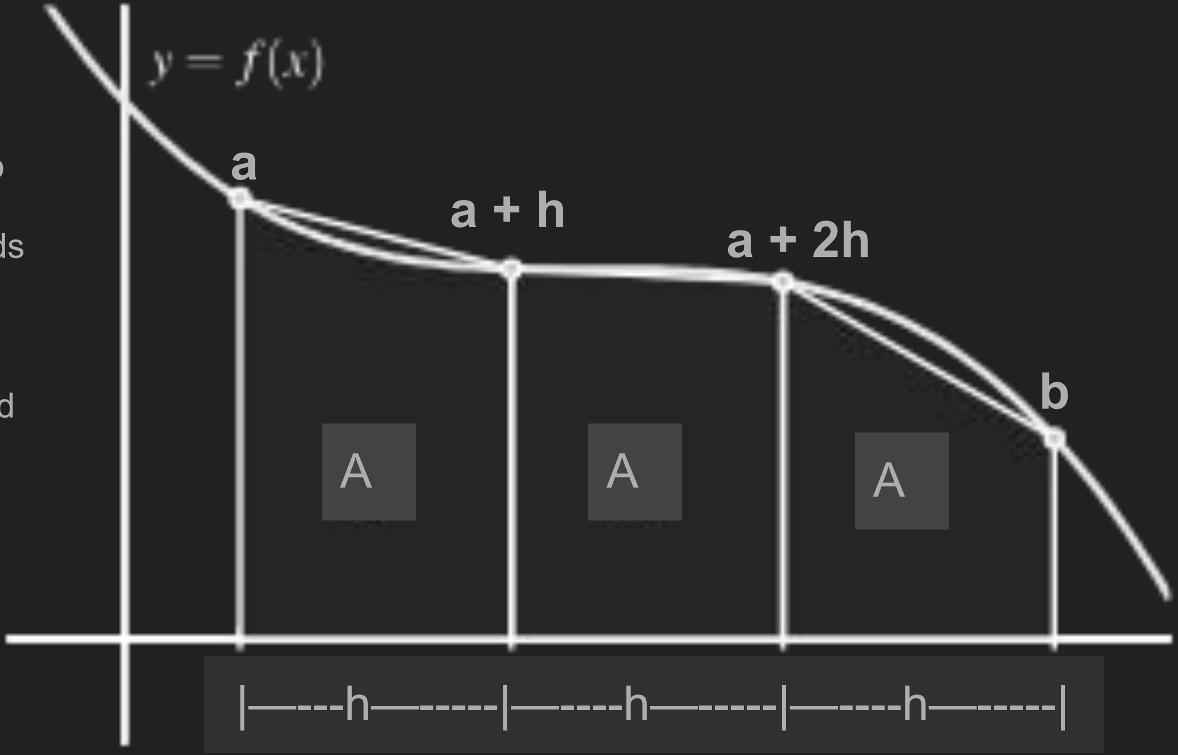
If we want to find the area between **a** and **b**

We can split this into equal length trapezoids

h = width of the trapezoid

We can calculate the area of each trapezoid

The area of f(x) ~ Sum of trapezoid areas

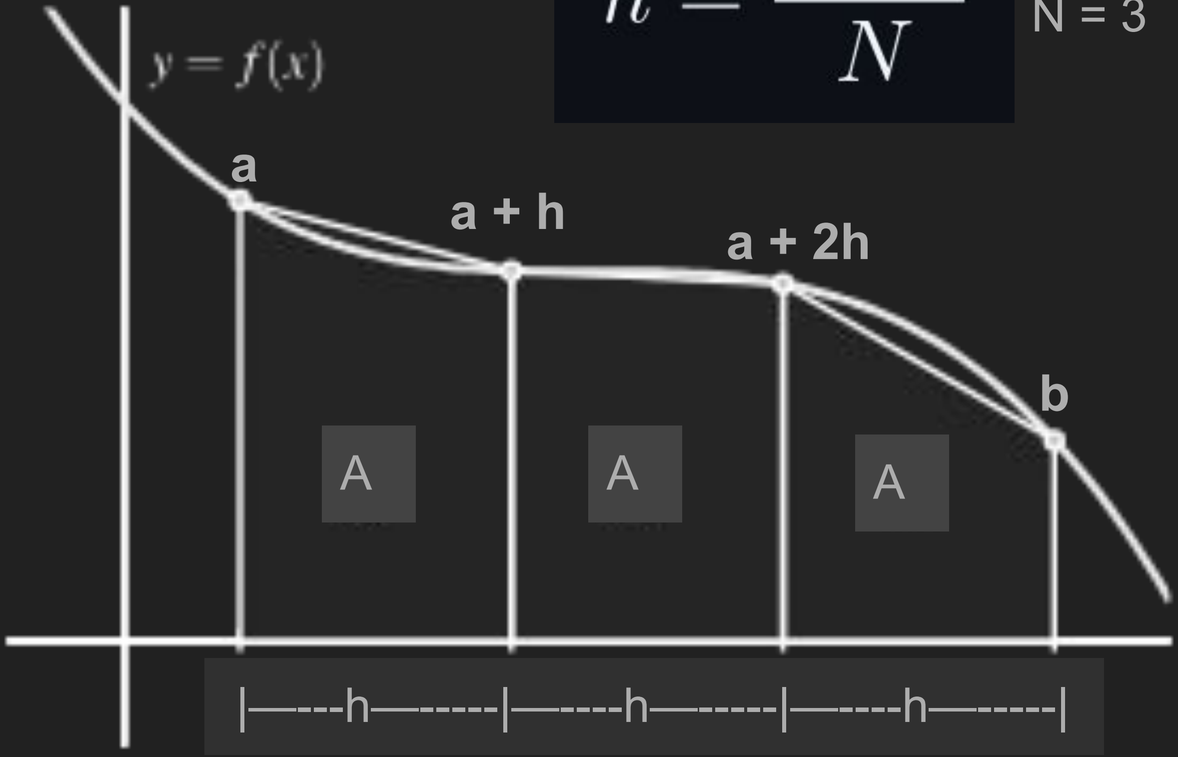


Trapezoidal rule

- Slight improvement to the rectangles (this is foreshadowing)
- Now for some Maths

$$h = \frac{b-a}{N}$$

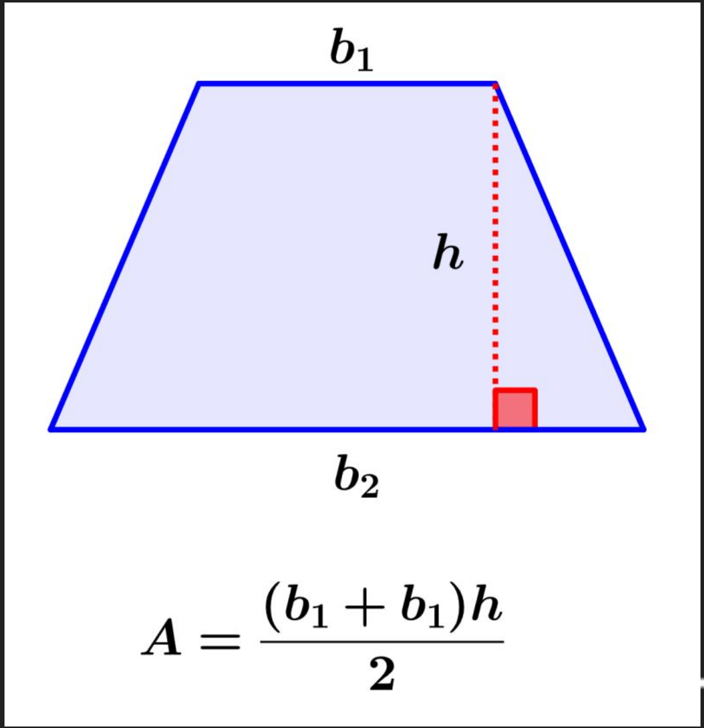
N = 3

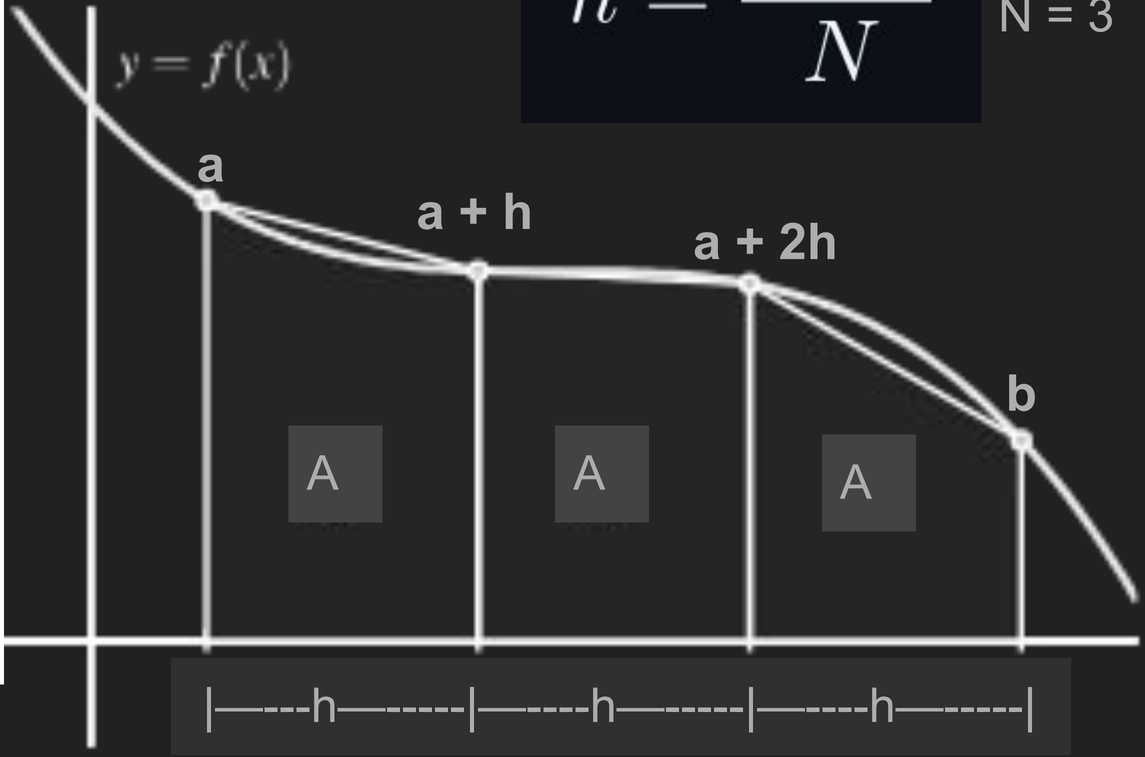


Trapezoidal rule

$$h = \frac{b-a}{N}$$

N = 3

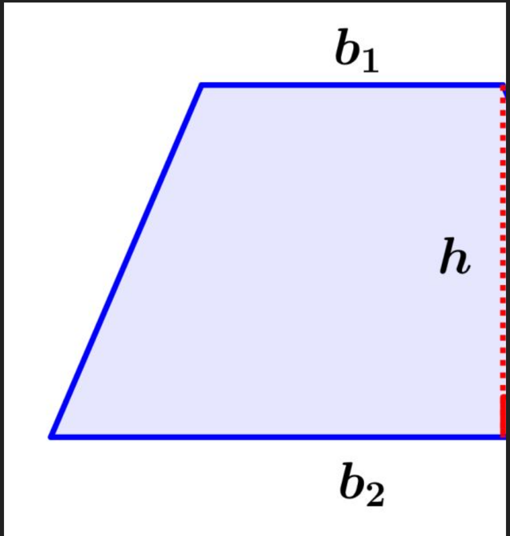




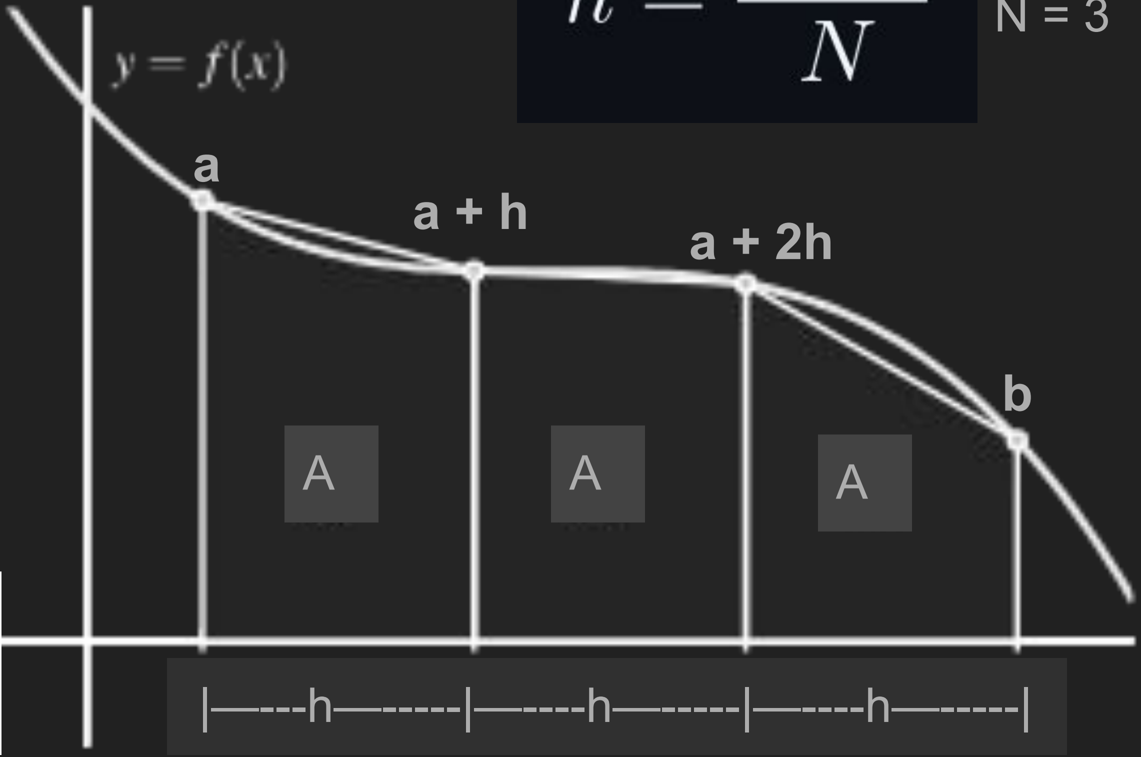
Trapezoidal rule

$$h = \frac{b - a}{N}$$

N = 3



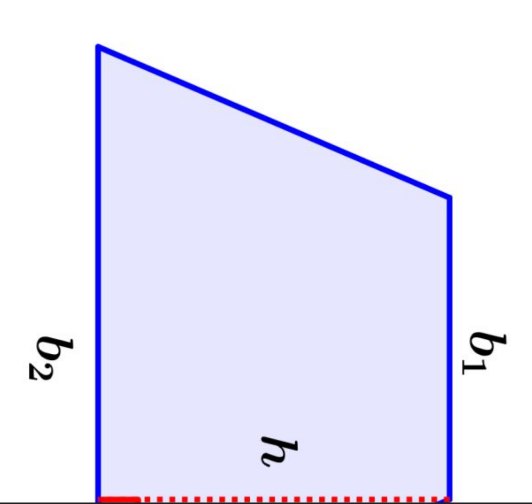
$$A = \frac{(b_1 + b_1)h}{2}$$

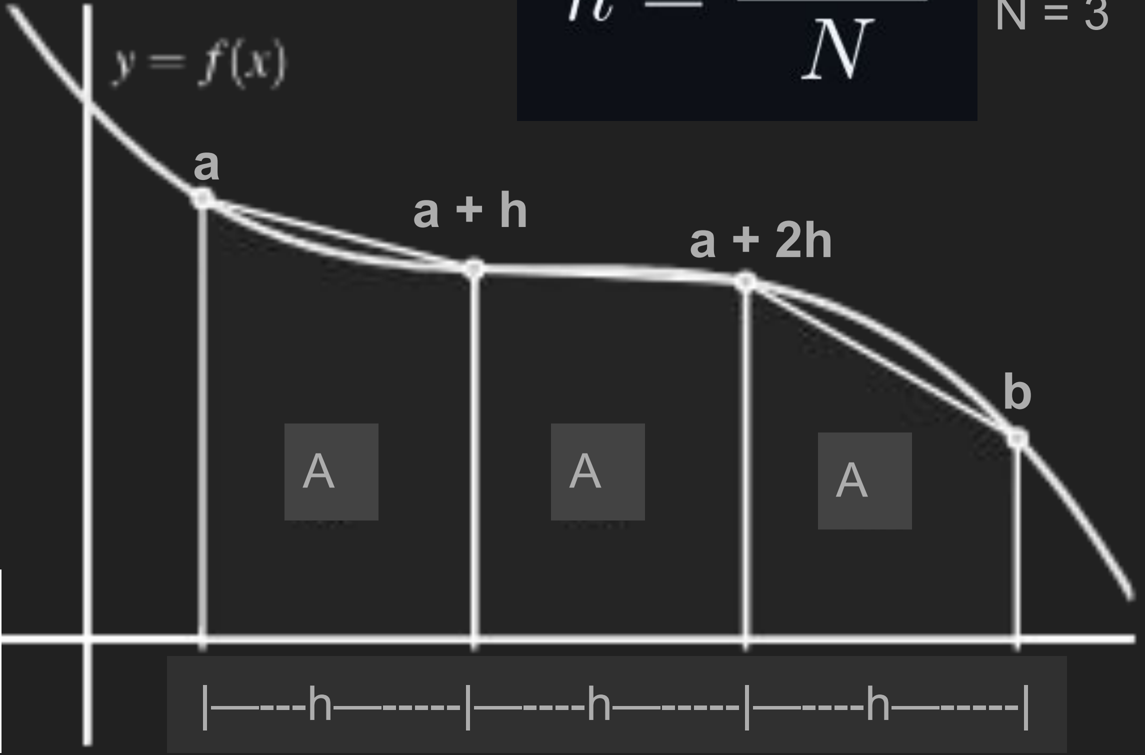


Trapezoidal rule

$$h = \frac{b-a}{N}$$

N = 3





$$A = \frac{(b_1 + b_1)h}{2}$$

Trapezoidal rule

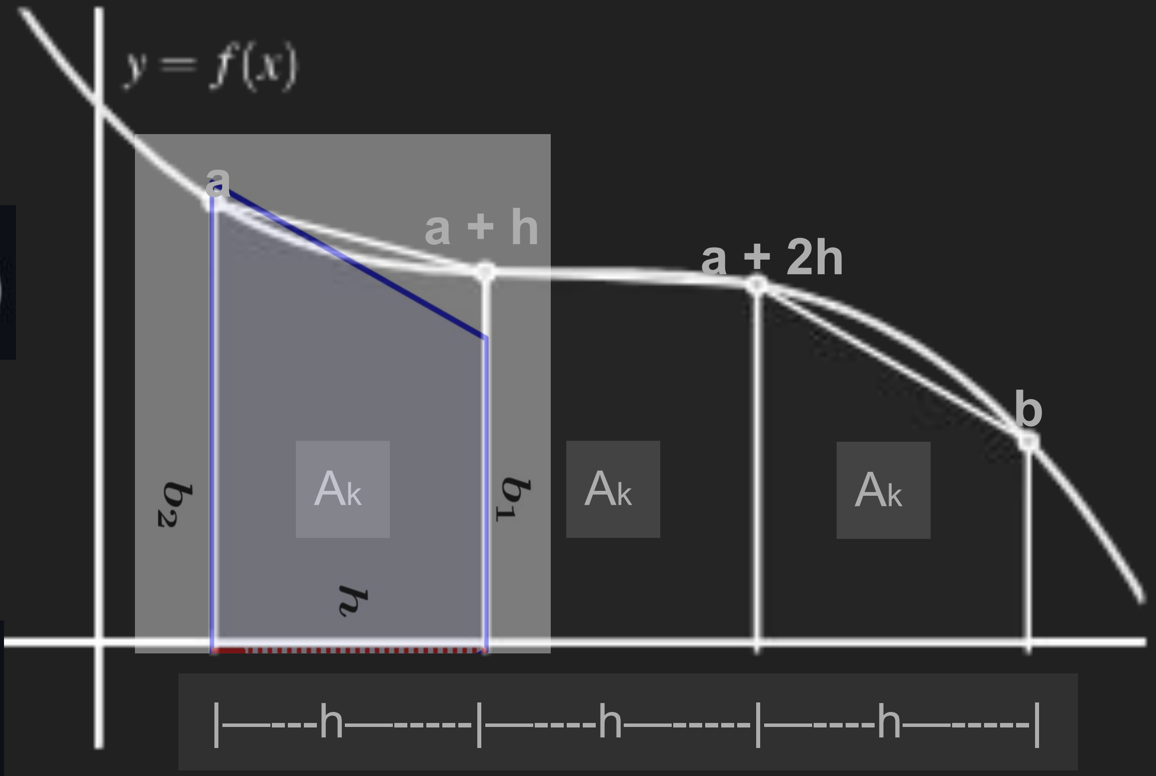
$$Area = \frac{1}{2}\ \text{height}\ \left[\text{length of two parallel lines added together}\right]$$

$$A = \frac{(b_1 + b_1)h}{2}$$

b1 = $f(a + h(k-1))$

b2 = $f(a + hk)$

$$A_k = \frac{1}{2}h\left[f(a+h(k-1)) + f(a+hk)\right]$$



Trapezoidal rule

Summing over all subintervals from $k = 1$ to N:

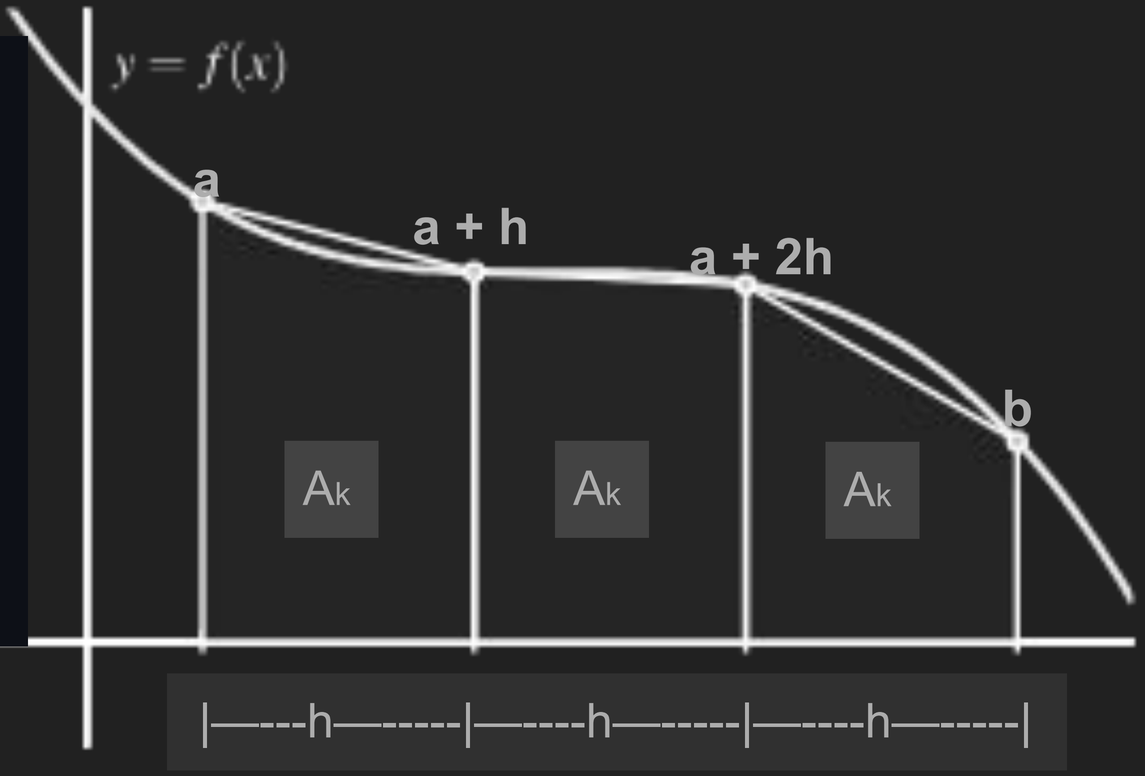
$$A \approx \frac{1}{2} h \sum_{k=1}^{N} \left[f(a + h(k-1)) + f(a + hk) \right]$$

Rewriting in a more compact form:

$$A \approx \frac{h}{2} \left[f(a) + f(b) + 2 \sum_{k=1}^{N-1} f(a + hk) \right]$$

...or as Mark prefers it:

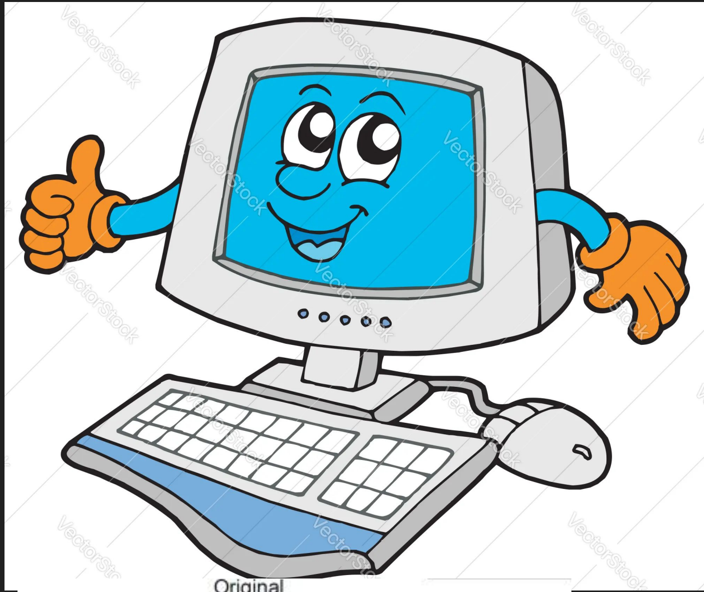
$$A \approx h \left[\frac{1}{2} f(a) + \frac{1}{2} f(b) + \sum_{k=1}^{N-1} f(a + hk) \right]$$

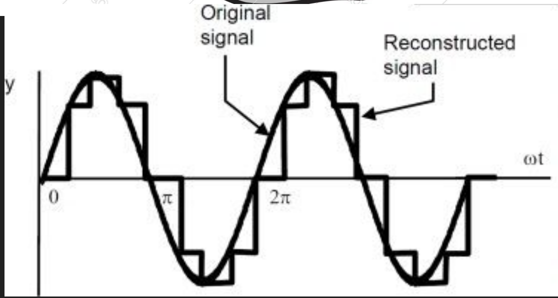


So if we integrate over data a computer has measured…

- N = len(x)

```python
def empirical_trapezoidal_rule(y_values, x_values, N):
    """
    Approximates the integral using trapezoidal rule for given y_values at given x_values.

    Parameters:
        y_values (array-like): The function values at given x points.
        x_values (array-like): The x values corresponding to y_values.
        N (int): Number of intervals.

    Returns:
        float: The approximated integral.
    """
    a, b = x_values[0], x_values[-1]
    h = x_values[1] - x_values[0]

    integral = (1/2) * (y_values[0] + y_values[-1]) * h  # First and last terms

    for k in range(1, N):
        xk = a + k * h  # Compute x_k explicitly
        yk = np.interp(xk, x_values, y_values)  # Interpolate y at x_k manually in loop
        integral += yk * h

    return integral


# Example usage with empirical data
x_data = np.array([0, 0.3, 0.6, 1])  # Given x data
y_data = np.array([1.0, 0.85, 0.55, 0.2])  # Corresponding y data
```





You can't learn to code from listening to me speak…

github.com/nialljmiller/PHYS4840_Numerical_integration_I

Download and complete “trapz.py”

How do we compute how wrong we are?

How wrong is the trapz method with N = 2, 10, 100?

Simpson’s rule 

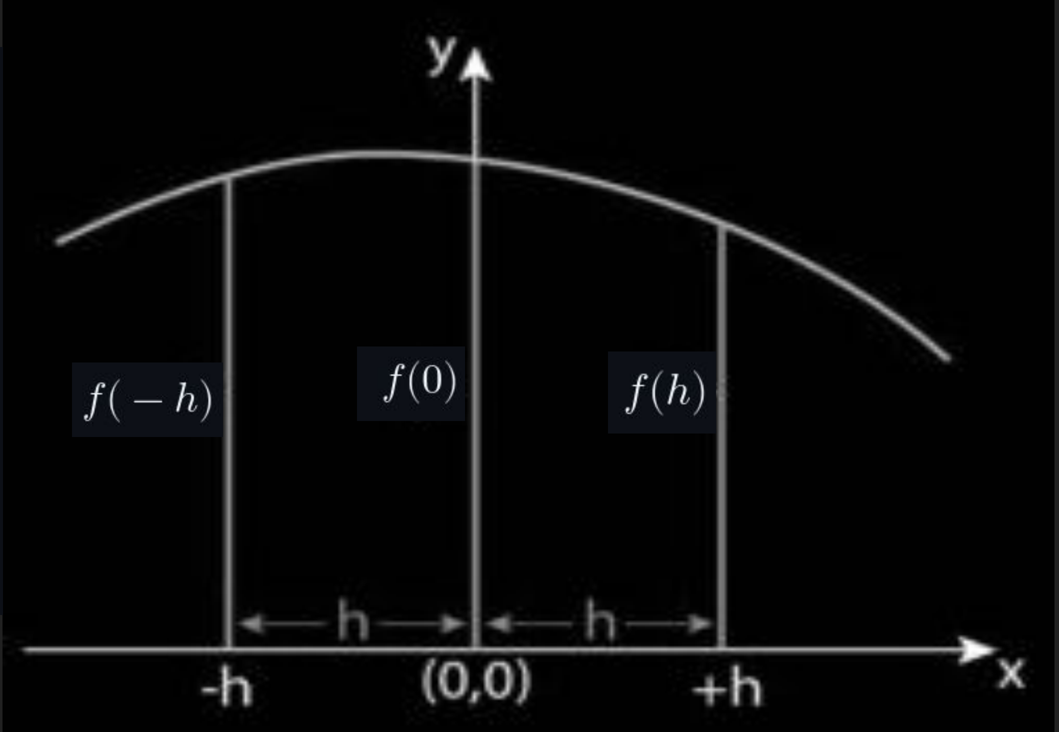
- Slight improvement to the trapezoids
- Now for some Maths

$$f(x) = Ax^2 + Bx + C$$

$$f(-h) = Ah^2 - Bh + C$$

$$f(0) = C$$

$$f(h) = Ah^2 + Bh + C$$



Simpson's rule 

- Now watch this cool trick

$$f(-h) = Ah^2 - Bh + C$$

$$f(0) = C$$

$$f(h) = Ah^2 + Bh + C$$

$$A = \frac{1}{2h^2}(f(h) - 2f(0) + f(-h))$$

$$B = \frac{1}{2h}(f(h) - f(-h))$$

$$C = f(0)$$

Simpson's rule 

- Now watch this cool trick

$$f(-h) = Ah^2 - Bh + C$$

$$f(0) = C$$

$$f(h) = Ah^2 + Bh + C$$

simultaneous equations

$$A = \frac{1}{2h^2}(f(h) - 2f(0) + f(-h))$$

$$B = \frac{1}{2h}(f(h) - f(-h))$$

$$C = f(0)$$

Simpson's rule 

- Now watch this cool trick

$$A = \frac{1}{2h^2}(f(h) - 2f(0) + f(-h))$$

$$B = \frac{1}{2h}(f(h) - f(-h))$$

$$C = f(0)$$

$$\int_{-h}^{h} (Ax^2 + Bx + C)dx$$

$$\frac{2}{3}Ah^3 + 2Ch$$

$$\frac{1}{3}h[f(-h) + 4f(0) + f(h)]$$

Simpson’s rule 

- Now watch this cool trick

$$A = \frac{1}{2h^2}(f(h) - 2f(0) + f(-h))$$

$$\cancel{B = \frac{1}{2h}(f(h) - f(-h))}$$

$$C = f(0)$$

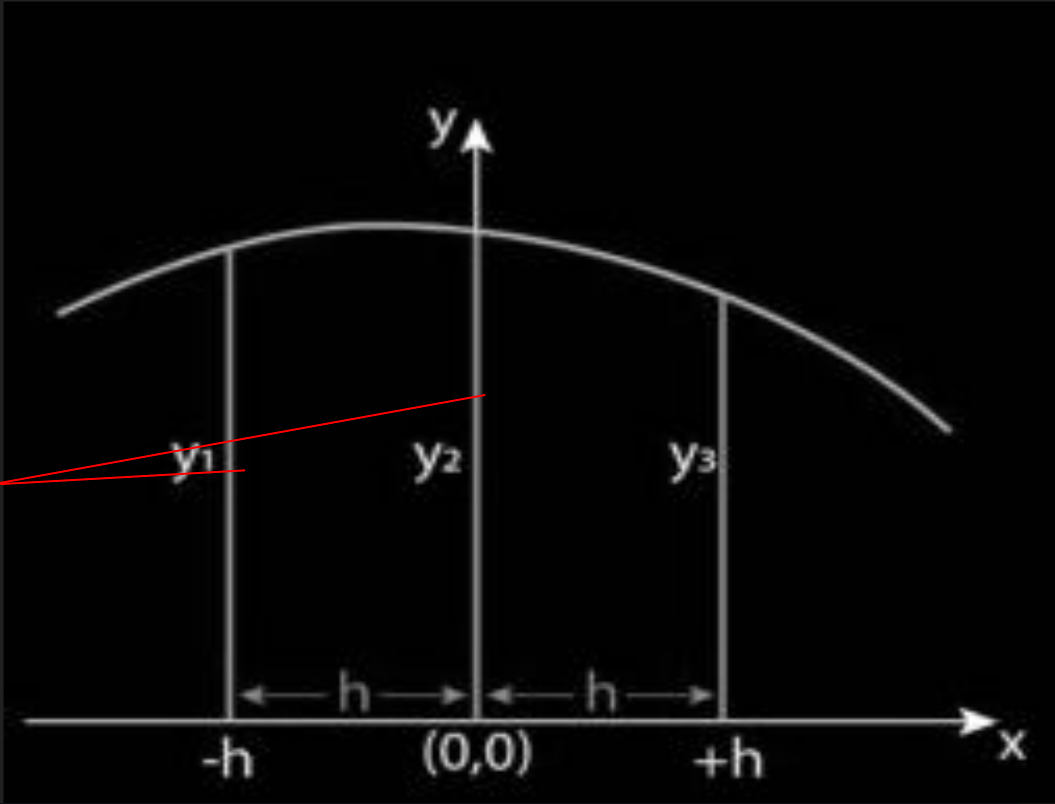
$$\int_{-h}^{h} (Ax^2 + Bx + C)dx$$

$$\frac{2}{3}Ah^3 + 2Ch$$

$$\frac{1}{3}h[f(-h) + 4f(0) + f(h)]$$

Simpson’s rule

$$\frac{1}{3}h[f(-h)+4f(0)+f(h)]$$



$$\int_a^b f(x), dx \approx \frac{h}{3}\left[f(a)+f(b)+4\sum_{\text{odd } k}^{1\ldots N-1} f(a+kh)+2\sum_{\text{even } k}^{2\ldots N-2} f(a+kh)\right]$$

Same as before…

github.com/nialljmiller/PHYS4840_Numerical_integration_I

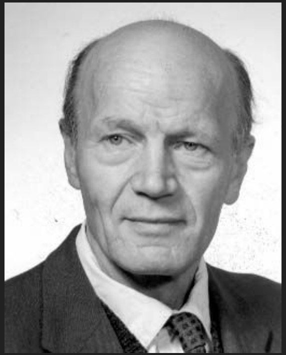
Download and complete “simpson.py”

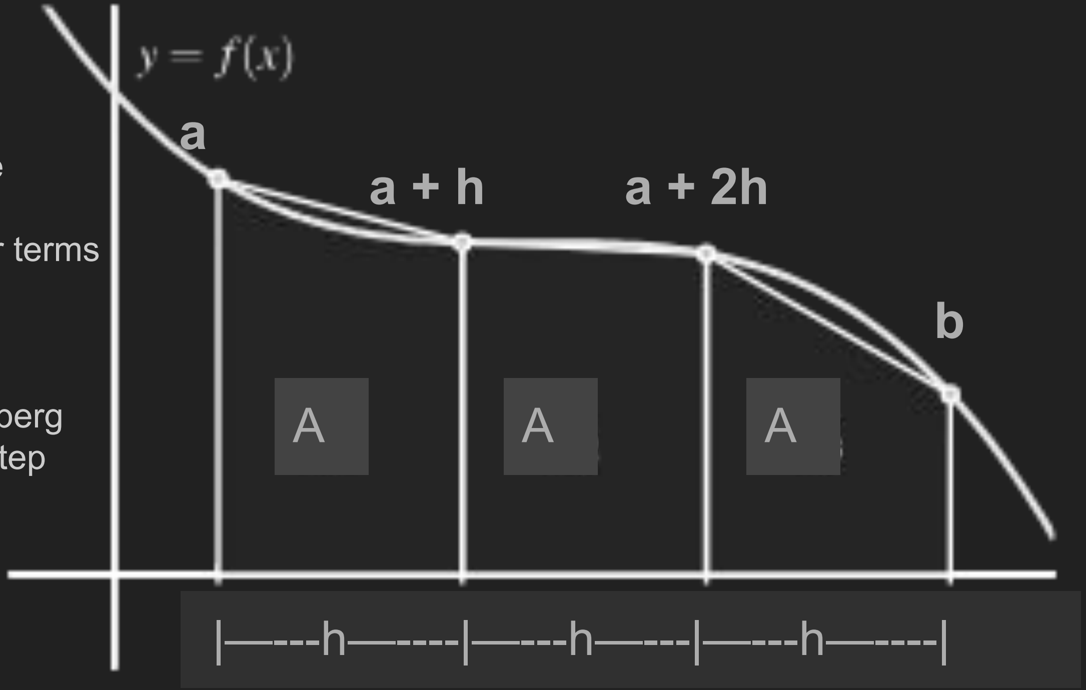
How wrong is the simpson method with N = 2, 10, 100?

How does this compare to trapz?

Romberg Integration (1955)

- Improvement to the trapezoids
- We do some fancy maths to extrapolate
- This generally outperforms the others



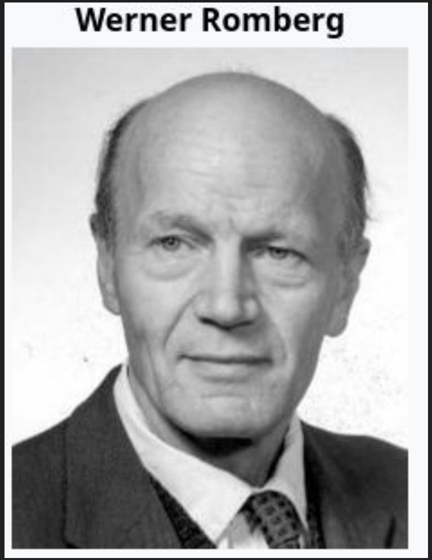


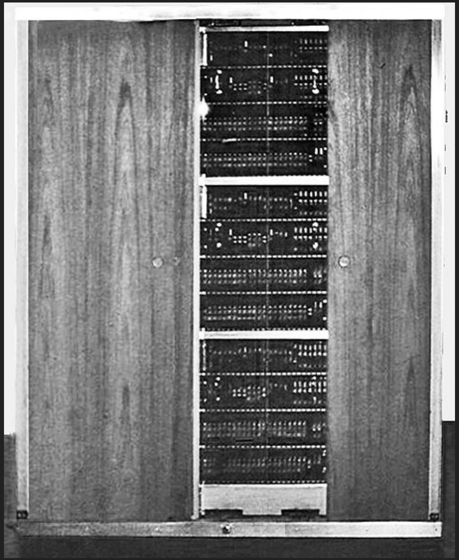
The **Romberg Integration** method builds upon the **Trapezoidal Rule** by applying **Richardson Extrapolation**, which systematically removes error terms to produce a more accurate result.

Instead of computing a single approximation, Romberg Integration refines the integral estimation step by step using a sequence of trapezoidal approximations.

My wikipedia detour - by Dr Miller (Math break)

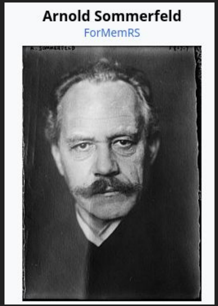
Werner Romberg was a German “half-jew” in the 1940s he fled to uppsala sweden, after WW2 he set up in trondheim Norway and went on to further numerical analysis and was instrumental in the installment of the first computer at trondheim.





My wikipedia detour - by Dr Miller (Math break)

What was his academic pedigree you ask?



My wikipedia detour - by Dr Miller (Math break)

What was his academic pedigree you ask?

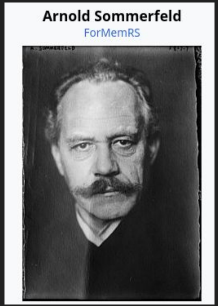
Werner Romberg

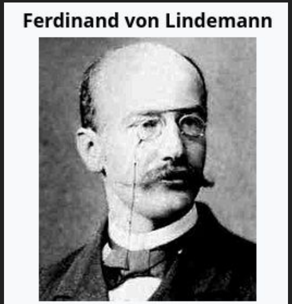
Arnold Sommerfeld
ForMemRS

Ferdinand von Lindemann

My wikipedia detour - by Dr Miller (Math break)

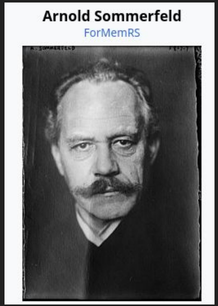
What was his academic pedigree you ask?

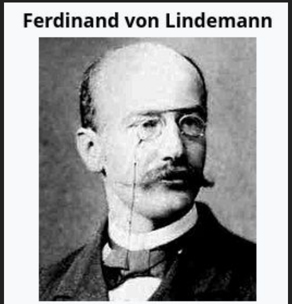


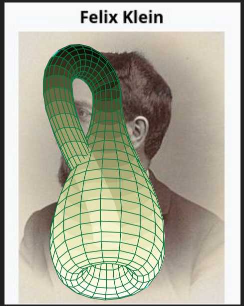


My wikipedia detour - by Dr Miller (Math break)

What was his academic pedigree you ask?

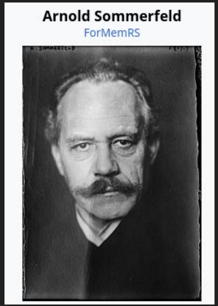


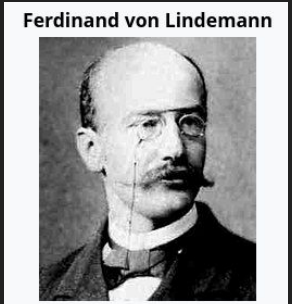


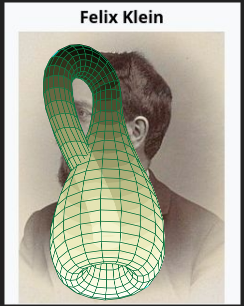


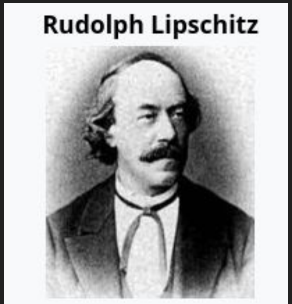
My wikipedia detour - by Dr Miller (Math break)

What was his academic pedigree you ask?









My wikipedia detour - by Dr Miller (Math break)

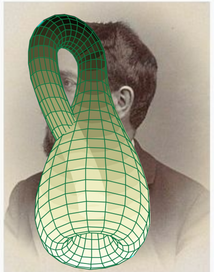
What was his academic pedigree you ask?

Werner Romberg

Arnold Sommerfeld
ForMemRS

Ferdinand von Lindemann

Felix Klein



Rudolph Lipschitz

Peter Gustav Lejeune Dirichlet

Siméon Poisson
FRS FRSE

My wikipedia detour - by Dr Miller (Math break)

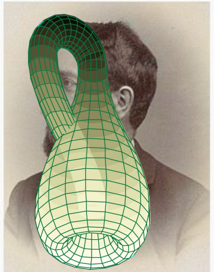
What was his academic pedigree you ask?

Werner Romberg

Arnold Sommerfeld
ForMemRS

Ferdinand von Lindemann

Felix Klein



Rudolph Lipschitz

Peter Gustav Lejeune Dirichlet

Siméon Poisson
FRS FRSE

Joseph Fourier

My wikipedia detour - by Dr Miller (Math break)

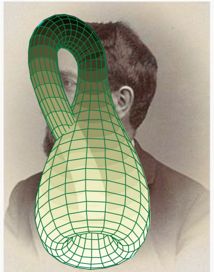
What was his academic pedigree you ask?

Werner Romberg

Arnold Sommerfeld
ForMemRS

Ferdinand von Lindemann

Felix Klein



Rudolph Lipschitz

Peter Gustav Lejeune Dirichlet

Siméon Poisson
FRS FRSE

Carl Friedrich Gauss

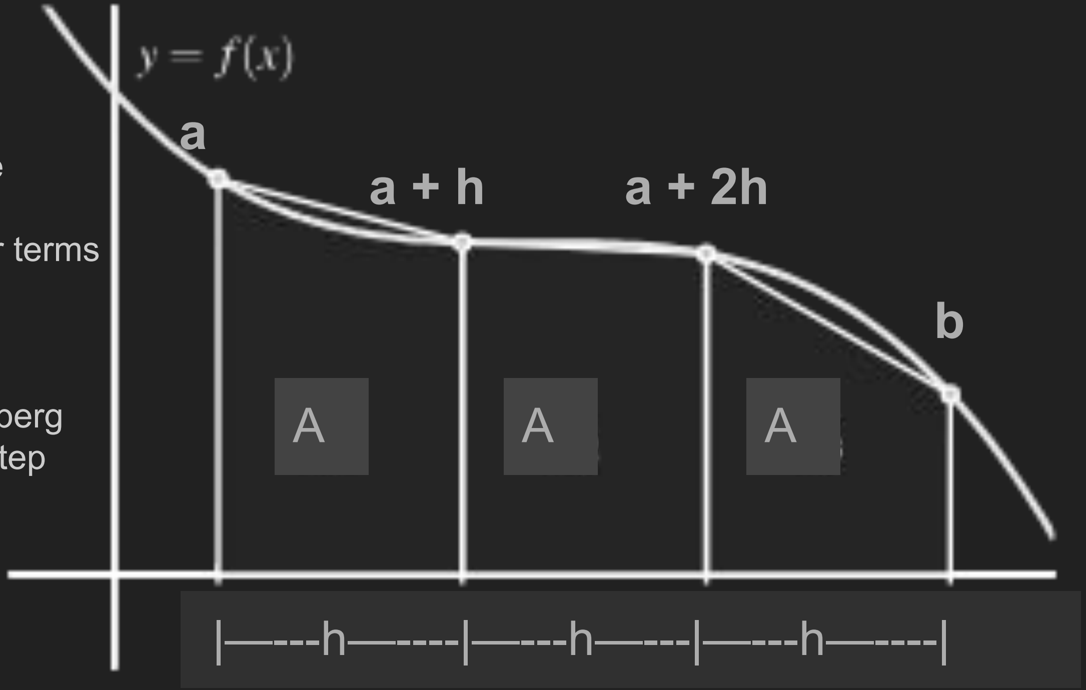
Joseph Fourier

Romberg Integration

- Slight improvement to the trapezoids
- We do some fancy maths to extrapolate
- This generally outperforms the others

The **Romberg Integration** method builds upon the **Trapezoidal Rule** by applying **Richardson Extrapolation**, which systematically removes error terms to produce a more accurate result.

Instead of computing a single approximation, Romberg Integration refines the integral estimation step by step using a sequence of trapezoidal approximations.



Romberg Integration

- Slight improvement to the trapezoids
- We do some fancy maths to extrapolate
- This generally outperforms the others

Instead of computing a single approximation, Romberg Integration refines the integral estimation step by step using a sequence of trapezoidal approximations.

See github for derivation

$$R_{m,n} = \frac{4^n R_{m,n-1} - R_{m-1,n-1}}{4^n - 1}$$

Romberg Integration Table

The Romberg method fills in a triangular table as follows:

(m)	($R_{m,0}$)	($R_{m,1}$)	($R_{m,2}$)	($R_{m,3}$)	...
0	(T_0)	-	-	-	-
1	(T_1)	($R_{1,1}$)	-	-	-
2	(T_2)	($R_{2,1}$)	($R_{2,2}$)	-	-
3	(T_3)	($R_{3,1}$)	($R_{3,2}$)	($R_{3,3}$)	-
...	...	...	...	...	...

Nested functions detour

“functionA” uses arg ‘a’ and passes args ‘b’ and ‘c’ to “functionB”.

“functionB” does not care about the argument names.

“functionA” must have been given ‘b’ and ‘c’ to be able to give them to “functionB”

```
def functionA(a, b, c):
    value = a
    value = value + functionB(b, c) + functionC(y = b, x = a)
    return value

#function B does not care what the arguments are called, but it does care about the order.
def functionB(x, y):
    value = x * y
    return value
```

Nested functions detour

```
def functionA(a, b, c):
    value = a
    value = value + functionB(b, c) + functionC(y = b, x = a)
    return value

#function B does not care what the arguments are called, but it does care about the order.
def functionB(x, y):
    value = x * y
    return value

#function C does care -- this is the difference between args and kwargs
def functionC(x = 'a', y = 'b'):
    return x + y

# Oops, we forgot to pass any arguments :(
result = functionA()
print(result)
```

Nested functions detour

```
def functionA(a, b, c):
    value = a
    value = value + f
    return value

#function B does not                              care about the order.
def functionB(x, y):
    value = x * y
    return value

#function C does care                             gs
def functionC(x = 'a
    return x + y

# Oops, we forgot to pass any arguments :(
result = functionA()
print(result)
```

github.com/nialljmiller/PHYS4840_Numerical_integration_I

nested_functions.py

In-class Exercise:

github.com/nialljmiller/PHYS4840_Numerical_integration_I

Download and complete "**integral_test.py**"
(This will take longer than previous, consult the github/textbook)

Run the timing test in a loop and save the error, time, N

Make a plot showing (see plot_example.py) :

- the accuracy of each method vs N
- the accuracy of each method vs compute time

Numerical integration II

Dr Niall Miller

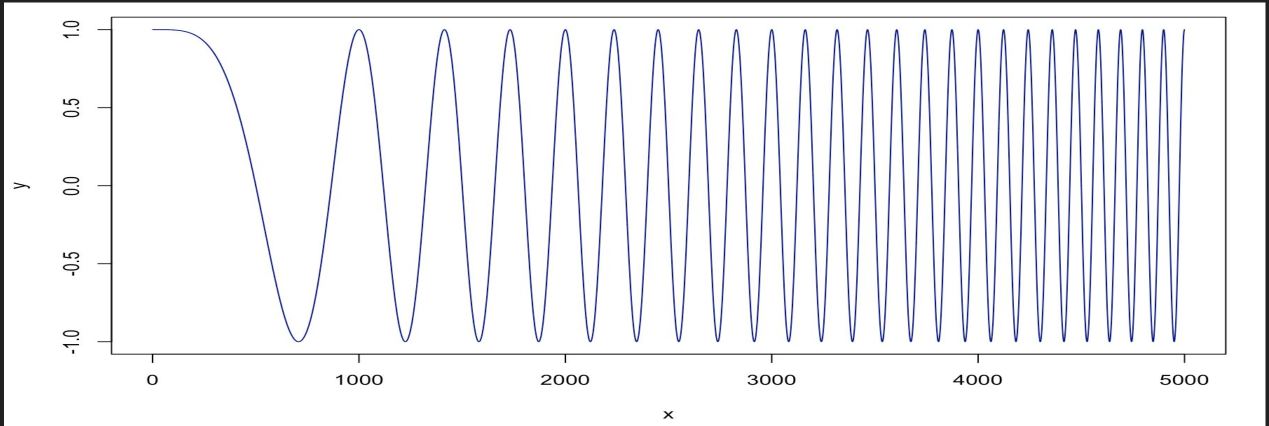
Gauss Legendre Quadrature

- Named as such because it predates computers
- Motivates by 19th century folk wanting to compute integrals for low N
  Simpsons rule for N = 1000 by hand is not fun

- Legendre polynomials are used to compute compute quaderature nodes and weights
- These ‘nodes’ are not determined as static. (i.e. not uniformly spaced)
- This allows this method to achieve high accuracy with fewer points as compared to the uniformly spaced method

An explanation of this method

- A limitation of the previously discussed methods is that they assume the ‘complexity’ of our function/distribution is consistent across the integrated region
- Does a consistent width size make sense for this?



- Instead of using equally spaced points (like in Trapezoidal or Simpson’s Rule), we choose points that make the approximation as accurate as possible for the least effort.
- The best points to use turn out to be the roots of Legendre polynomials.

Gauss-Legendre

As usual, we approximate an integral with a sum.
This time a weighted sum of the form:

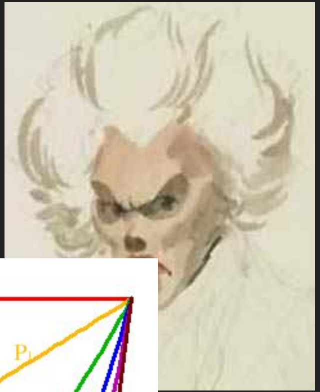
$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i\, f(x_i)$$

So all this method is:

- Consult legendre polynomials to find our weights and roots
- Calculate the value of our function at each of our Legendre ‘roots’
- Multiply the value by the weight
- Sum everything together
- Done

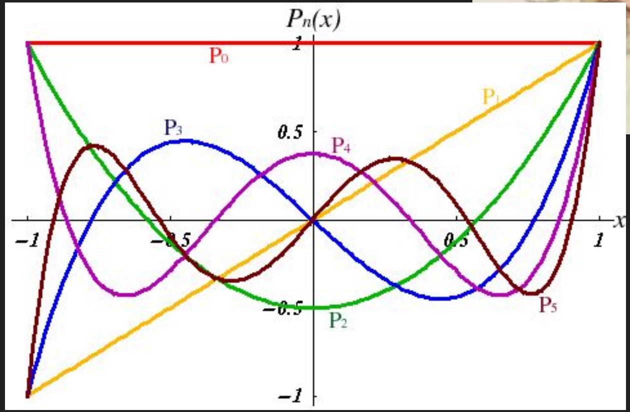
Legendre polynomials?

- These are polynomials $P_n(x)$ that appear in many areas of mathematics and physics.



$$(1-x^2)P_n''(x) - 2xP_n'(x) + n(n+1)P_n(x) = 0.$$

$$\int_{-1}^{1} P_m(x)P_n(x)\,dx = 0 \quad \text{if } n \neq m.$$

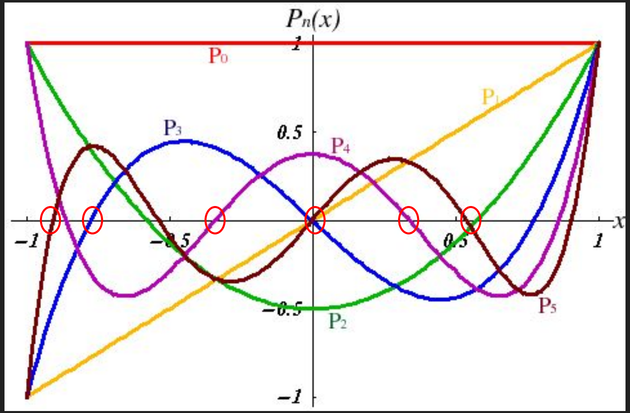


- The key point we need to know is that we use the root values
  This means we take the value of x where **$P_n(x)$ = 0**
- The Legendre polynomials will give use the **positions** and **weights** for our integral

Legendre polynomials?

- These are polynomials $P_n(x)$ that appear in many areas of mathematics and physics.

$$P_0(x) = 1$$
$$P_1(x) = x$$
$$P_2(x) = \tfrac{1}{2}(3x^2 - 1)$$
$$P_3(x) = \tfrac{1}{2}(5x^3 - 3x)$$
$$P_4(x) = \tfrac{1}{8}(35x^4 - 30x^2 + 3)$$
$$P_5(x) = \tfrac{1}{8}(63x^5 - 70x^3 + 15x)$$
$$P_6(x) = \tfrac{1}{16}(231x^6 - 315x^4 + 105x^2 - 5).$$



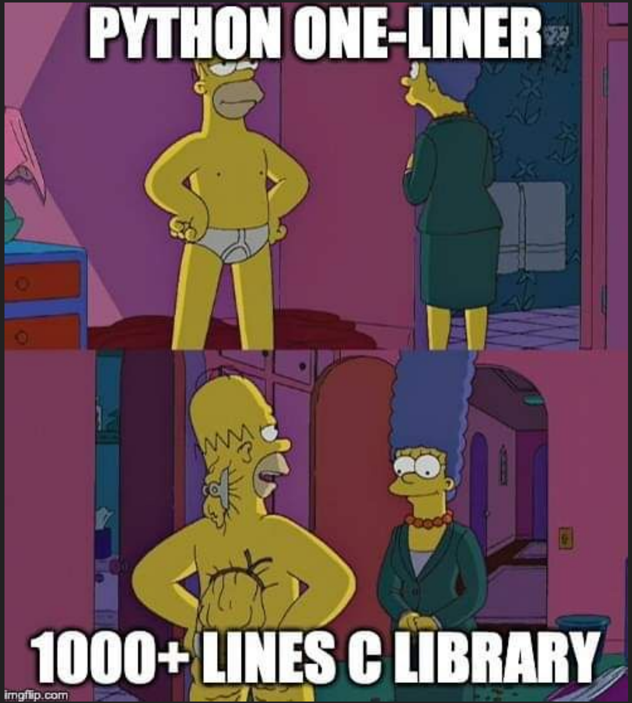
- The key point we need to know is that we use the root values
  This means we take the value of x where $\mathbf{P_n(x) = 0}$
- The Legendre polynomials will give use the **positions** and **weights** for our integral

That sounds easy …apart from Legendre

Numpy is amazing!

np.polynomial.legendre.leggauss(deg)

“Computes the sample points and weights for Gauss-Legendre quadrature.”



That sounds easy …apart from Legende

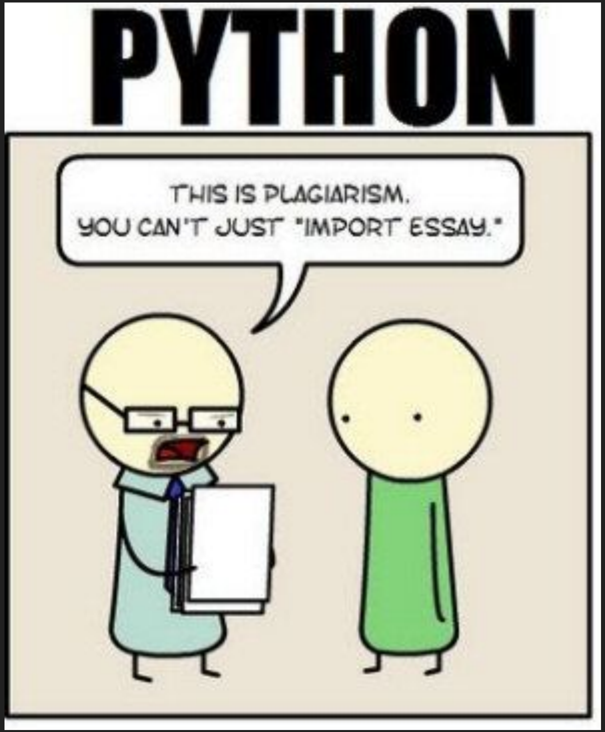
NumPy reference > Routines and objects by topic > Polynomials > Legendre Series (`numpy.polynomial.legendre`) > numpy.polynomial.legendre.leggauss

numpy.polynomial.legendre.leggaus

`polynomial.legendre.leggauss(deg)` [source]

Gauss-Legendre quadrature.

Computes the sample points and weights for Gauss-Legendre quadrature. These sample points and weights will correctly integrate polynomials of degree $2 * deg - 1$ or less over the interval $[-1, 1]$ with the weight function $f(x) = 1$.



"Computes the sample points and weights for Gauss-Legendre quadrature."

That sounds easy …apart from Legendre

np.polynomial.legendre.leggauss(deg)

$$P_0(x) = 1$$
$$P_1(x) = x$$
$$P_2(x) = \frac{1}{2}(3x^2 - 1)$$
$$P_3(x) = \frac{1}{2}(5x^3 - 3x)$$
$$P_4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3)$$
$$P_5(x) = \frac{1}{8}(63x^5 - 70x^3 + 15x)$$
$$P_6(x) = \frac{1}{16}(231x^6 - 315x^4 + 105x^2 - 5).$$

It literally just solves for x in the case that $P_n(x) = 0$

```
#how to do thing in python
import thing_doer as thingy

thing = thingy.do_thing()
```

Python thingys!

```python
from numpy.polynomial.legendre import leggauss as legendre_thingy

legendre_roots, weights = legendre_thingy(3)
print(legendre_roots, weights)
```

```
njm@fedora [13:21:35] [~/PHYS4840_Numerical_integration_I] [main]
-> % python python.py
Gauss-Legendre Quadrature Points (Roots of P_n(x)):
[-0.77459667  0.          0.77459667]

Weights for each point:
[0.55555556 0.88888889 0.55555556]
```

Lets code together!

github.com/nialljmiller/PHYS4840_Numerical_integration_I

gauss_legendre.py

In-class Exercise:

github.com/nialljmiller/PHYS4840_Numerical_integration_I

Download and complete "**integral_test.py**"
(This will take longer than previous, consult the github/textbook)

Run the timing test in a loop and save the error, time, N

Make a plot showing (see plot_example.py) :

- the accuracy of each method vs N
- the accuracy of each method vs compute time