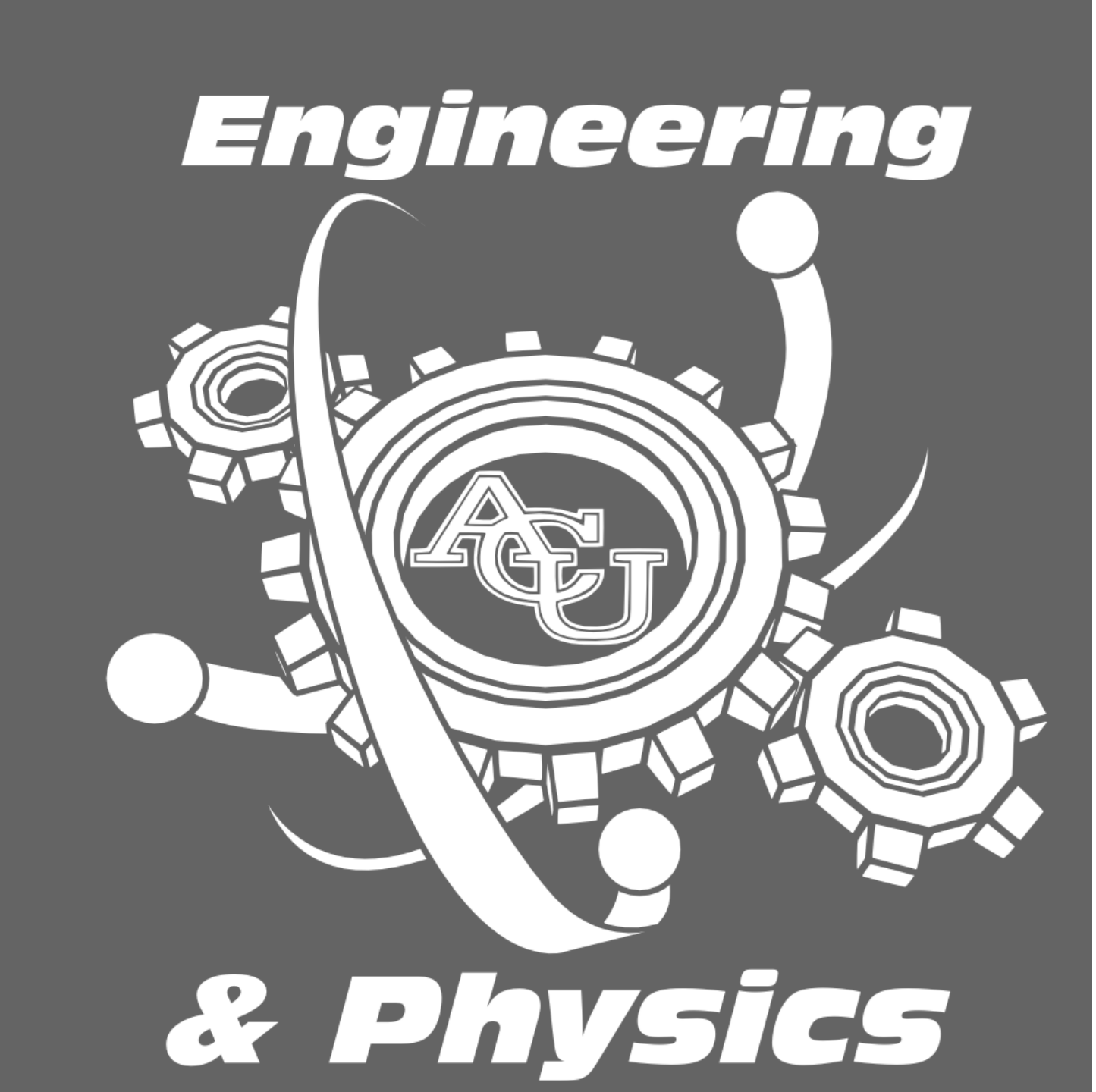


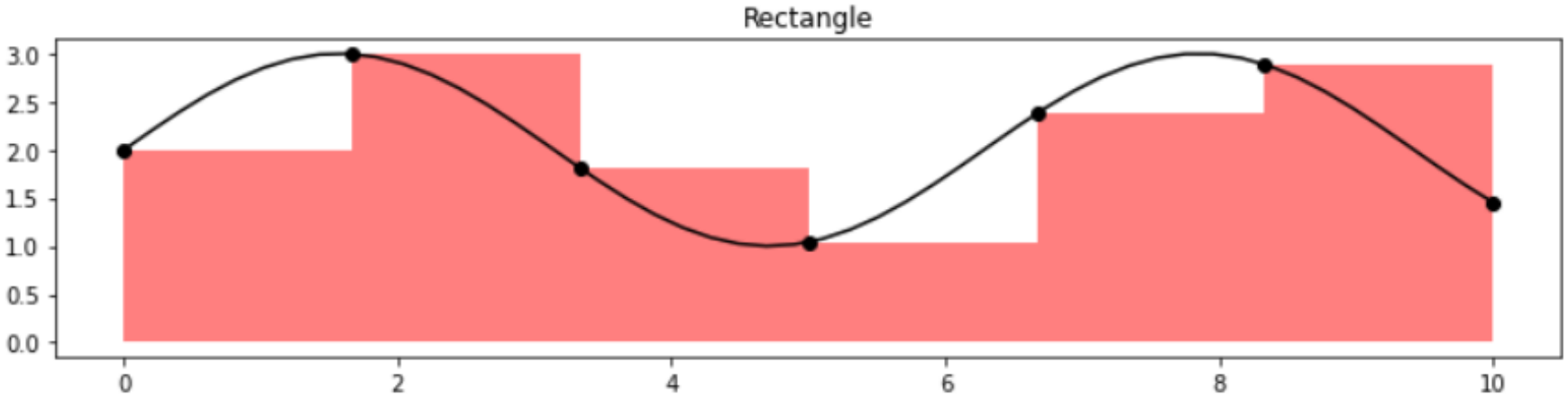
PHYS 351

Integrals

Dr. Daugherity

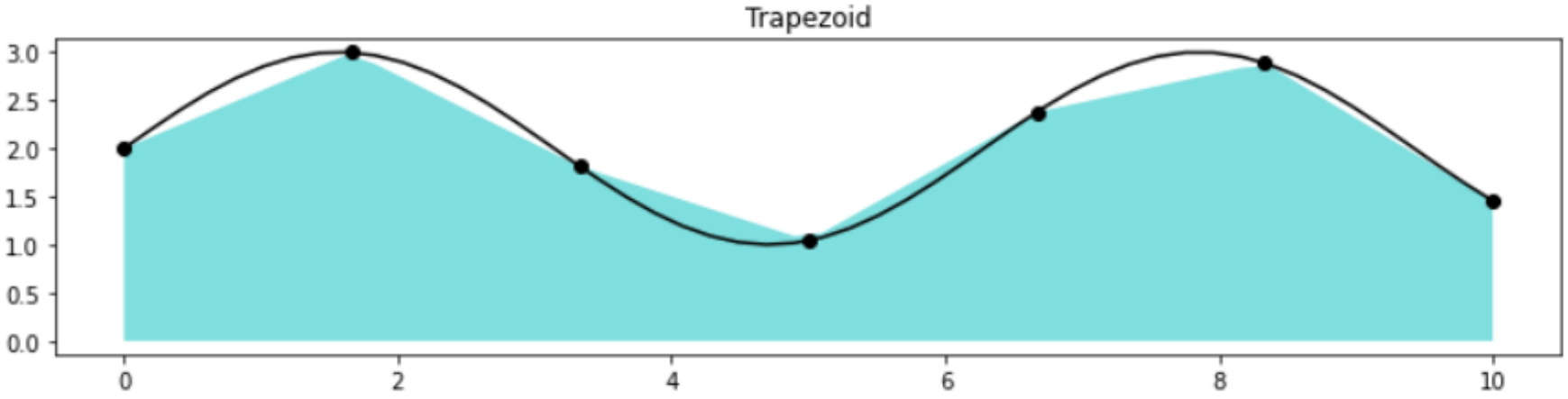
Abilene Christian University

DISCRETE DATA POINTS

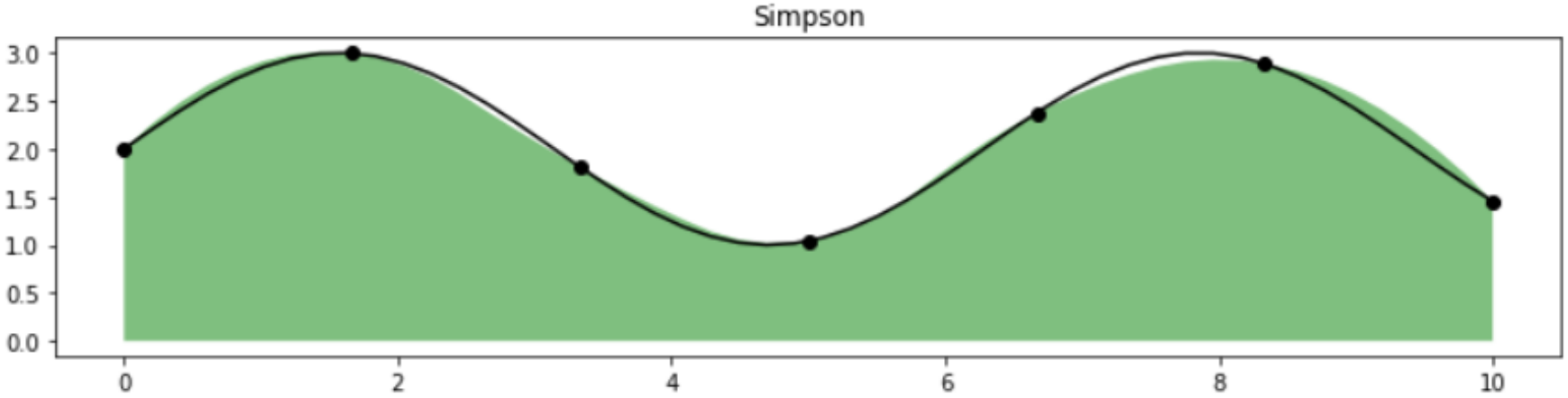


$$I = \sum w_i \cdot y_i$$

$$w = h[1,1,1,\ldots,0]$$



$$w = \frac{h}{2}[1,2,\ldots,2,1]$$



$$w = \frac{h}{3}[1,4,2,4,2,\ldots,4,1]$$

odd # points, even # panels

Richardson Extrapolation

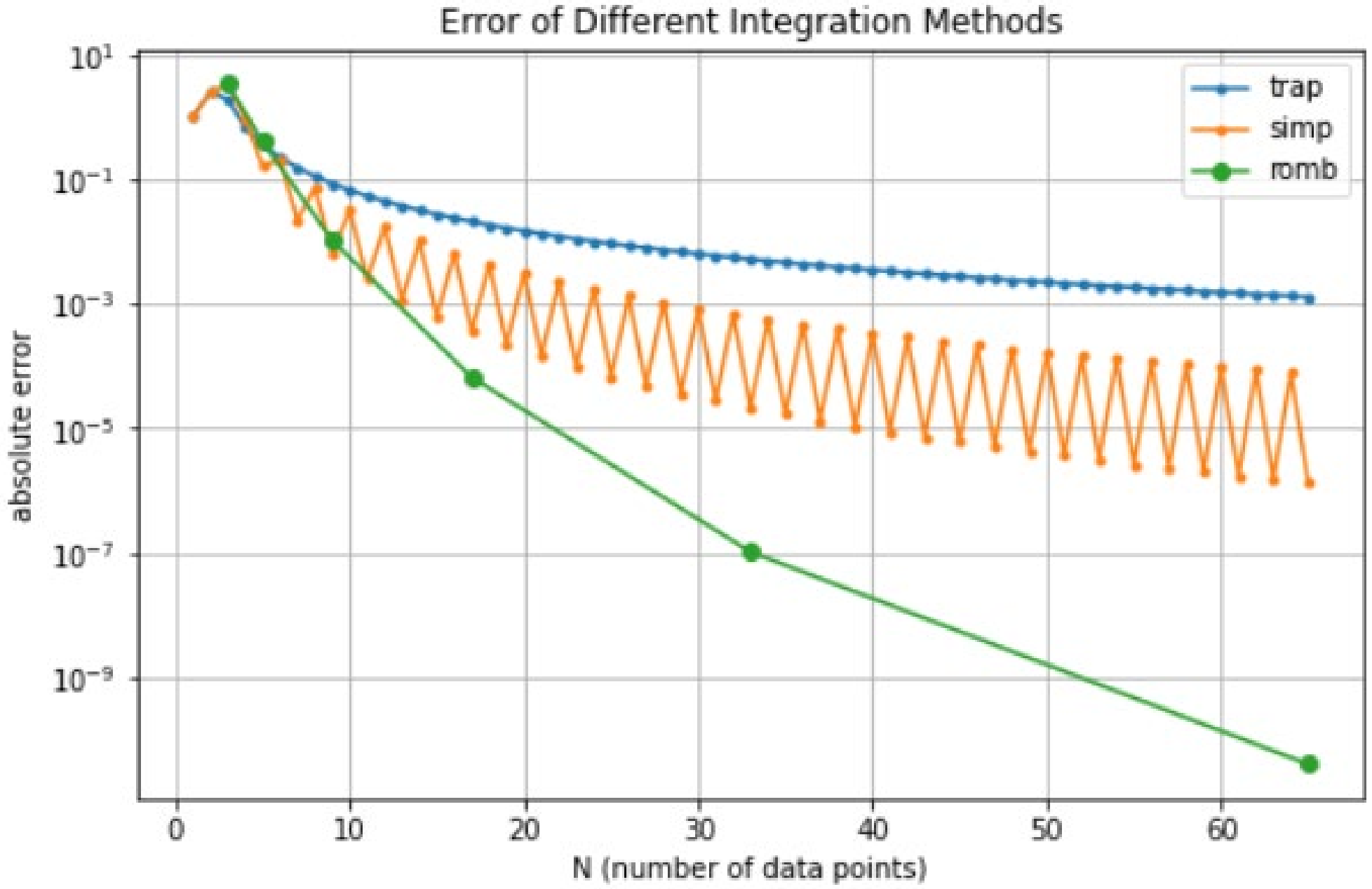
Assume: $R = A(h) + E(h)$ with $E(h) \approx ch^p + \cdots$

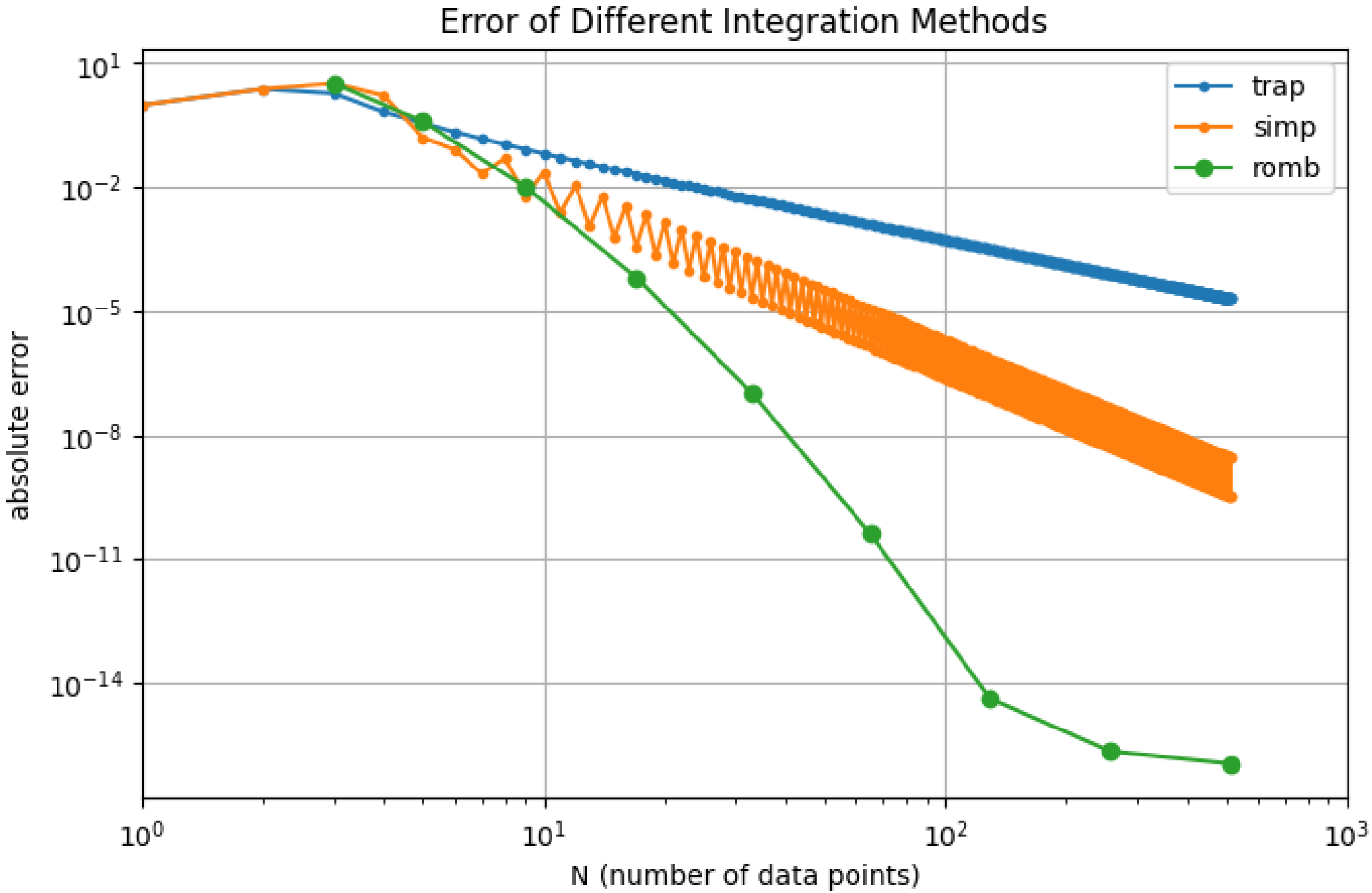
Combine to cancel leading error:

$$R = \frac{2^p\ A(h) - A(2h)}{2^p - 1}$$

Romberg

- Repeatedly combines trapezoid integrals with different # of points in a special way that cancels out the leading error term (Richardson extrapolation)
- Requires $2^k + 1$ evenly spaced points


Error of Different Integration Methods
trap
simp
romb
absolute error
10^{1}
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
0
10
20
30
40
50
60
N (number of data points)


Error of Different Integration Methods
trap
simp
romb
absolute error
N (number of data points)
10^{1}
10^{-2}
10^{-5}
10^{-8}
10^{-11}
10^{-14}
10^{0}
10^{1}
10^{2}
10^{3}

Trapezoid API

https://numpy.org/doc/1.25/reference/generated/numpy.trapz.html

numpy.trapz

`numpy.trapz(y, x=None, dx=1.0, axis=-1)` [source]

Integrate along the given axis using the composite trapezoidal rule.

If *x* is provided, the integration happens in sequence along its elements - they are not sorted.

Integrate *y* (*x*) along each 1d slice on the given axis, compute $\int y(x)dx$. When *x* is specified, this integrates along the parametric curve, computing $\int_t y(t)dt = \int_t y(t)\frac{dx}{dt}\Big|_{x=x(t)} dt$.

Parameters:

y : *array_like*

Input array to integrate.

x : *array_like, optional*

The sample points corresponding to the *y* values. If *x* is None, the sample points are assumed to be evenly spaced *dx* apart. The default is None.

dx : *scalar, optional*

The spacing between sample points when *x* is None. The default is 1.

Same as scipy.integrate.trapezoid

Simpson API

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.simpson.html#scipy.integrate.simpson

simpson(*y*, *, *x*=*None*, *dx*=*1.0*, *axis*=*-1*) [source]

Integrate y(x) using samples along the given axis and the composite Simpson's rule. If x is None, spacing of dx is assumed.

If there are an even number of samples, N, then there are an odd number of intervals (N-1), but Simpson's rule requires an even number of intervals. The parameter 'even' controls how this is handled.

Parameters:

y : *array_like*

Array to be integrated.

x : *array_like, optional*

If given, the points at which *y* is sampled.

dx : *float, optional*

Spacing of integration points along axis of *x*. Only used when *x* is None. Default is 1.

- must include "x=" when calling this to avoid a warning
- There used to be a parameter called "even" that was removed in Aug 2024

Romberg API

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.romb.html

scipy.integrate.

romb

romb(*y*, *dx=1.0*, *axis=-1*, *show=False*) [source]

Romberg integration using samples of a function.

Parameters:

y : ***array_like***

A vector of `2**k + 1` equally-spaced samples of a function.

dx : ***float, optional***

The sample spacing. Default is 1.

axis : ***int, optional***

The axis along which to integrate. Default is -1 (last axis).

show : ***bool, optional***

When *y* is a single 1-D array, then if this argument is True print the table showing Richardson extrapolation from the samples. Default is False.

Since trapz and simpson allow for uneven spacing, they take the x array as a parameter. Romberg requires even spacing, so only dx (step size) is allowed

Discrete Data Points Summary

Method	Python Function	Requires evenly spaced points?	restrictions on N?	Error $\propto$
trapezoid	np.trapz	no	none	$\frac{(b-a)h^2}{12}$
Simpson	scipy.integrate.simpson	no	odd	$\frac{(b-a)h^4}{180}$
Romberg	scipy.integrate.romb	yes	2^k+1	varies

Challenge

- Pick a function where you know the exact integral
- How many data points do you need to get an error of 1e-6 using
  - np.trapz
  - scipy.integrate.simpson
  - scipy.integrate.romb

CONTINUOUS FUNCTIONS

Continuous Functions

- While we can always just sample the function at N evenly-spaced data points and use the previous methods, we can take advantage of our **continuous** functions and do whatever we want with the step sizes.
- We get to choose N data points, so we have 2N pieces of information
- Exact solution if $f(x)$ is degree $2N-1$ polynomial

METHODVS NOVA

INTEGRALIVM VALORES PER APPROXIMATIONEM INVENIENDI.

AVCTORE

CAROLO FRIDERICO GAVSS

SOCIETATI REGIAE SCIENTIARVM EXHIBITA D. 16. SEPT. 1814.

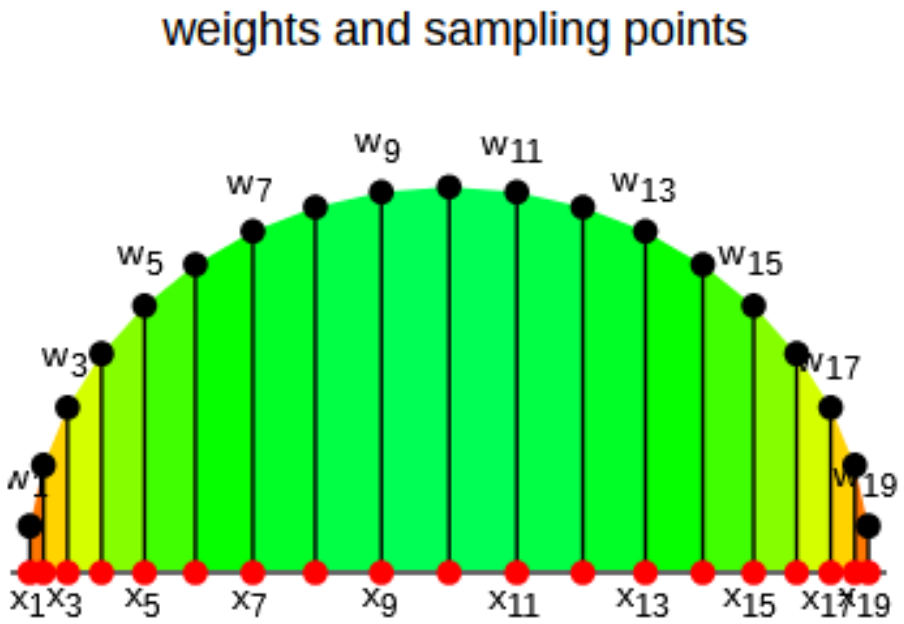
I.

Inter methodos ad determinationem numericam approximatam integralium propoſitas inſignem tenent locum regulae, quas praeeunte ſummo *Newton* euolutas dedit *Cotes*. Scilicet ſi requiritur valor integralis $\int y\,dx$ ab $x = g$ vsque ad $x = h$ ſumendus, valores ipſius y pro his valoribus extremis ipſius x et pro quotcunque aliis intermediis a primo ad vltimum incrementis aequalibus progredientibus, multiplicandi ſunt per certos coëfficientes numericos, quo facto productorum aggregatum in $h - g$ ductum integrale quaeſitum ſuppeditabit, eo maiore praeciſione, quo plures termini in hac operatione adhibentur. Quum principia huius methodi, quae a geometris rarius quam par est in vſum vocari videtur, nullibi quod ſciam plenius explicata ſint, pauca de his praemittere ab inſtituto noſtro haud alienum erit.

- Method introduced by Gauss in 1814. Modern formulation by Jacobi in 1826
- When everything was done by hand people were highly motivated to find efficient methods
- Reworked for computers in 1960’s, QUADPACK in 1983

Gaussian Quadrature

- Derivation is **EXTREMELY** complex
- Idea is to use a special weight function and evaluate the function at special points to minimize error



Number of points, n	Points, x_i		Weights, w_i	
1	0		2	
2	$\pm\frac{1}{\sqrt{3}}$	±0.57735...	1	
3	0		$\frac{8}{9}$	0.888889...
	$\pm\sqrt{\frac{3}{5}}$	±0.774597...	$\frac{5}{9}$	0.555556...
4	$\pm\sqrt{\frac{3}{7}-\frac{2}{7}\sqrt{\frac{6}{5}}}$	±0.339981...	$\frac{18+\sqrt{30}}{36}$	0.652145...
	$\pm\sqrt{\frac{3}{7}+\frac{2}{7}\sqrt{\frac{6}{5}}}$	±0.861136...	$\frac{18-\sqrt{30}}{36}$	0.347855...
5	0		$\frac{128}{225}$	0.568889...
	$\pm\frac{1}{3}\sqrt{5-2\sqrt{\frac{10}{7}}}$	±0.538469...	$\frac{322+13\sqrt{70}}{900}$	0.478629...
	$\pm\frac{1}{3}\sqrt{5+2\sqrt{\frac{10}{7}}}$	±0.90618...	$\frac{322-13\sqrt{70}}{900}$	0.236927...

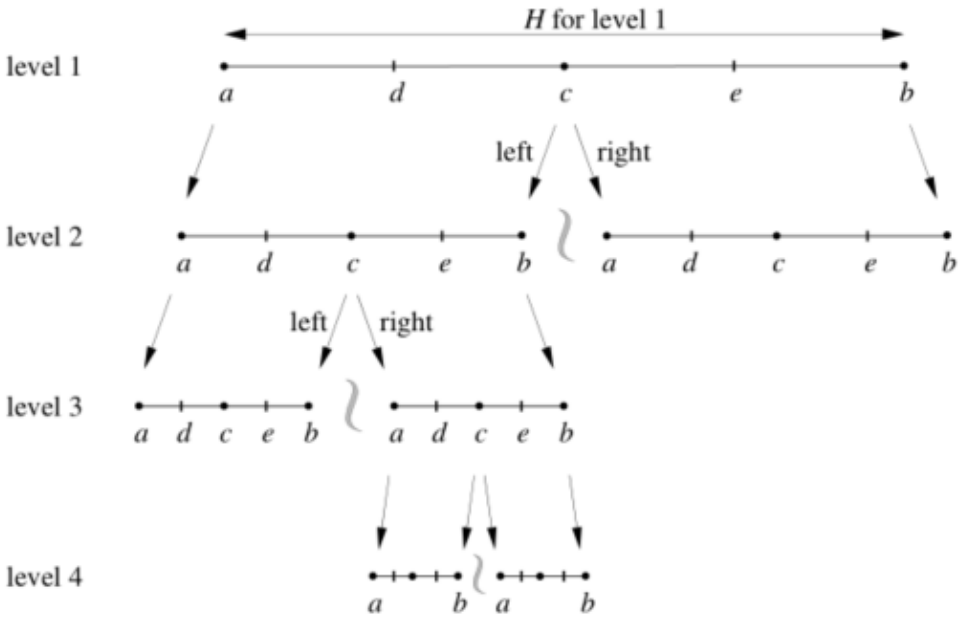
https://en.wikipedia.org/wiki/Gaussian_quadrature

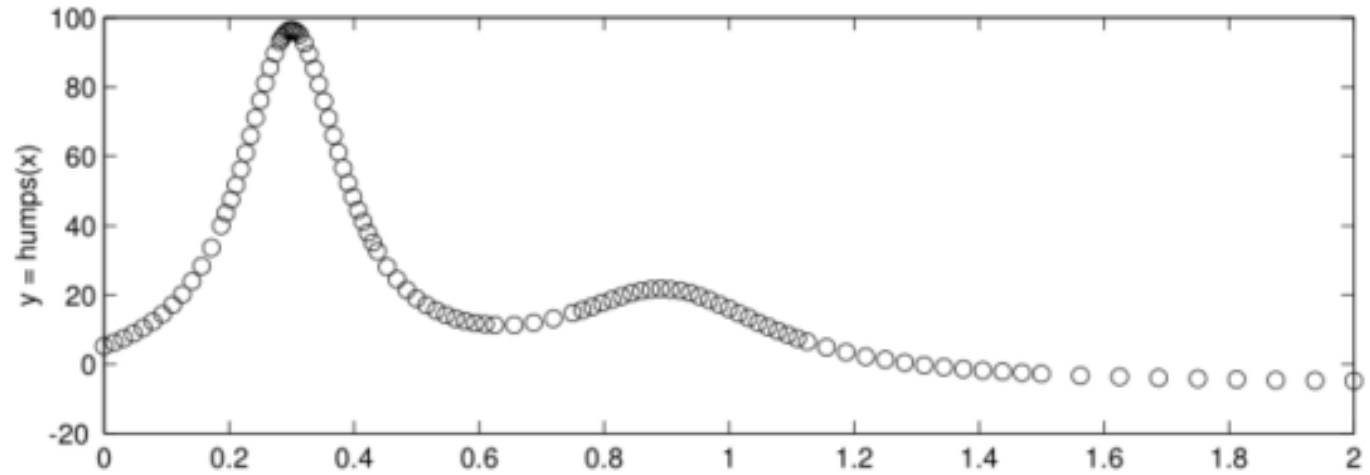
Adaptive Gaussian Quadrature

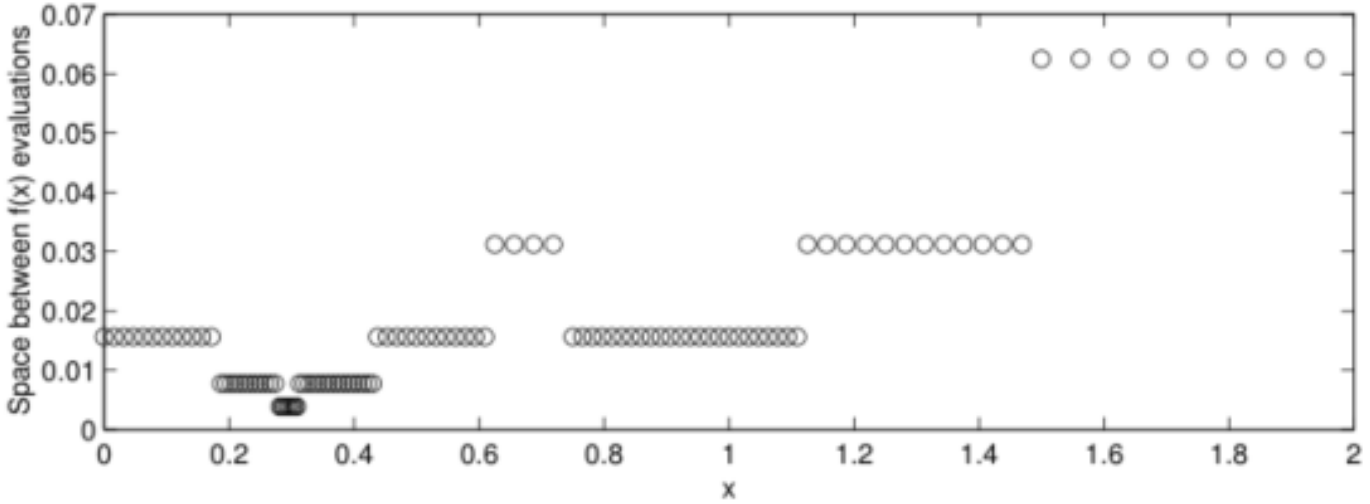
- We can estimate the error by seeing how much the function varies as we calculate the integral.
- If the error in any region gets too big, we can subdivide that region into smaller steps
- Extremely sophisticated algorithms for this were written in Fortran in the 1980's as the QUADPACK library. Any modern program uses these methods.
- To be clear: Integrating continuous functions is a solved problem, just use Adaptive Gaussian Quadrature.

Adaptive Gaussian Quadrature

Simplified Example







Gauss-Kronrod Quadrature

Scipy (and everything else) uses algorithms from QUADPACK. The strategy is called Gauss–Kronrod quadrature:

- start with n point Gaussian quadrature accurate to $2n-1$
- add $n+1$ new points with new weights so total is accurate to $3n+1$

By default QUADPACK:

1. starts with $n=10$ Gaussian ($G10$) with order 19
2. adds 11 new points for $n=21$ Kronrod ($K21$) with order 64
3. Error estimate is $K21-G10$. If error is too large split both regions in half and repeat.

Gauss-Kronrod Quadrature

```
1 def f(x):
2     return x**19
3
4 quad(f,-1,1,full_output=1)
```

```
(0.0,
 1.1102230246252337e-15,
 {'neval': 21,
  'last': 1,
```

```
1 def f(x):
2     return x**20
3
4 quad(f,-1,1,full_output=1)
```

```
(0.09523809523809527,
 3.335055349289531e-14,
 {'neval': 63,
  'last': 2,
```

Note: if you look at the full output of quad, only the first entries up to the value of **last** are defined, the other entries are random garbage. Thanks Fortran!

Things to Know

- Use `trapz`, `simpson`, or `romb` for data points (and know how these work)
- Use `scipy.integrate.quad` for continuous functions (Adaptive Gaussian Quadrature)
- Fitting / Spline Interpolation is also an option

Calculus Summary

	Continuous Function	Data Points
Derivative	Write your own centered difference or use other package	`numpy.gradient`
Integral	`scipy.integrate.quad`	`numpy.trapz`, `simpson` or `romb` from `scipy.integrate`