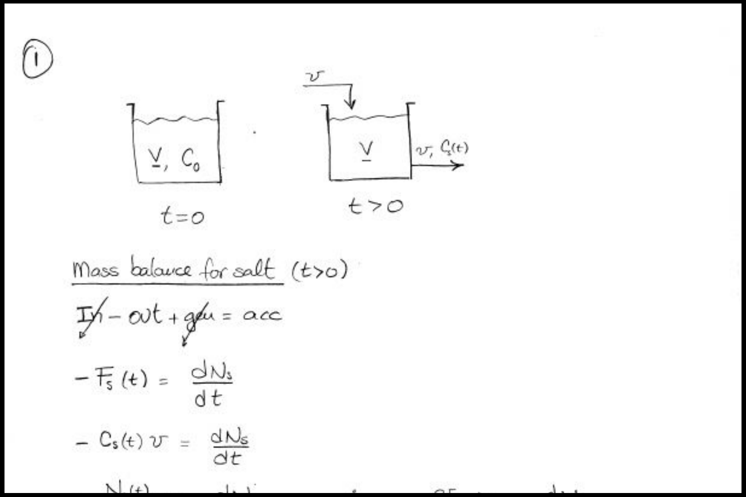
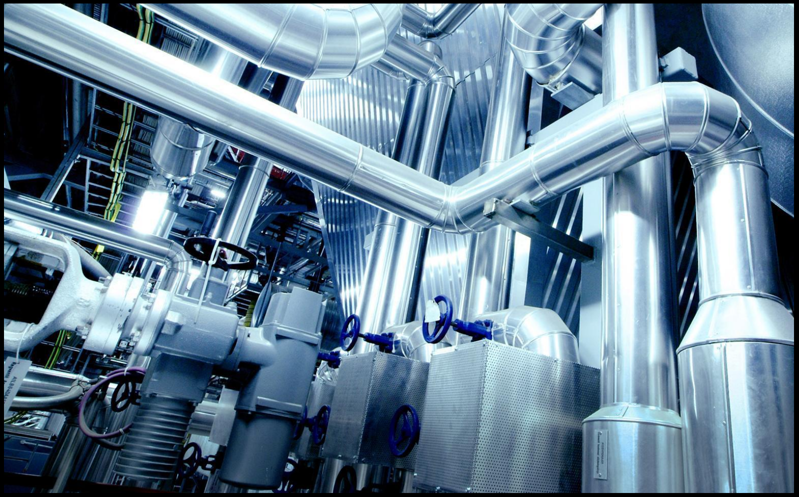




ChE352

Numerical Techniques for Chemical Engineers

Professor Stevenson

Lecture 8

A bright idea from NYU

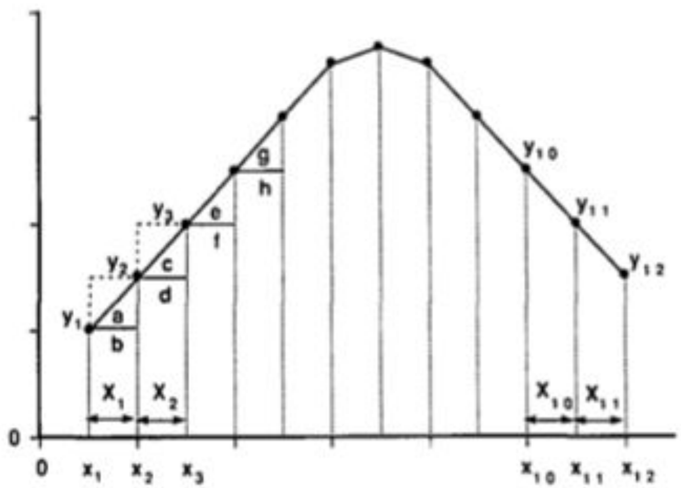


Figure 1—*Total area under the curve is the sum of individual areas of triangles a, c, e, and g and rectangles b, d, f, and h.*

A Mathematical Model for the Determination of Total Area Under Glucose Tolerance and Other Metabolic Curves
Journal: Diabetes Care, 1994

- In 1994, doctors at NYU's Department of Nutrition invented a method for finding area under a curve
- Allowed better treatment of diabetes patients

"The strategy of this mathematical model is to divide the total area under a curve into individual small segments such as squares, rect-angles, and triangles, whose areas can be precisely determined."

A bright idea from NYU

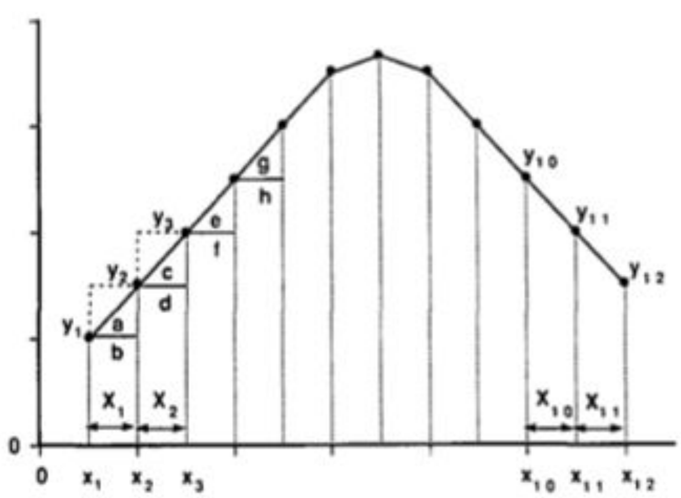


Figure 1—*Total area under the curve is the sum of individual areas of triangles a, c, e, and g and rectangles b, d, f, and h.*

A Mathematical Model for the Determination of Total Area Under Glucose Tolerance and Other Metabolic Curves
Journal: Diabetes Care, 1994

- In 1994, doctors at NYU's Department of Nutrition invented a method for finding area under a curve
- Allowed better treatment of diabetes patients
- The method was over 2000 years old at the time
- Better late than never!

Activity: define using limits

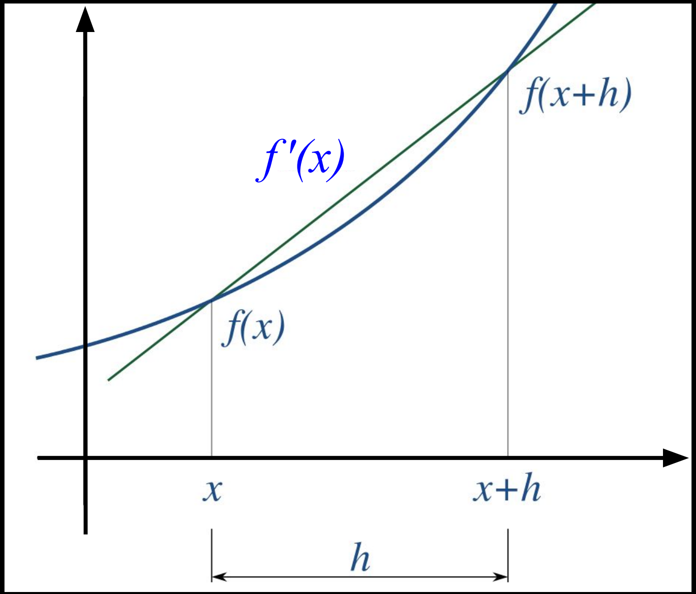
1. Write (from memory) the limit which defines the derivative of a function f(x) at a point x_0
   - Use h to represent the change in x
2. Write (from memory) the limit which defines the integral of a function f(x) over the interval [a,b]
   - Assume the interval is subdivided into n equally-spaced subintervals, each of size h
   - Also give the value of h in terms of a, b, & n

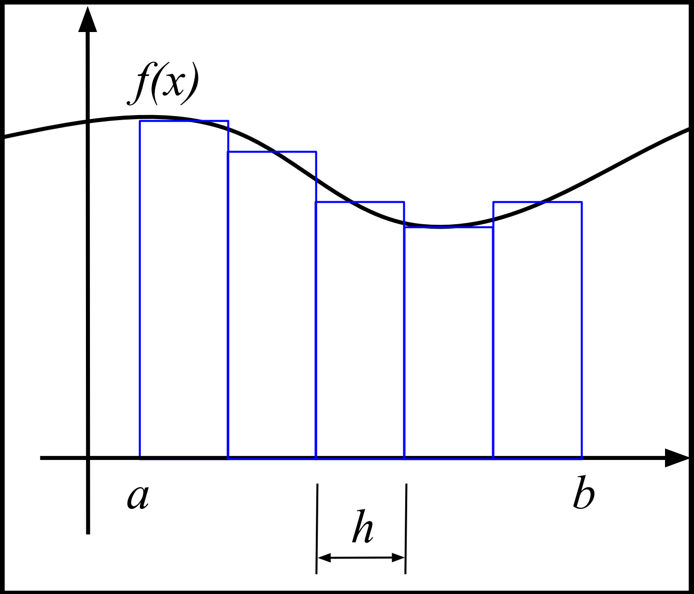
Definition of derivative & integral

$$f'(x_0) = \lim_{h\to 0} \frac{f(x_0+h) - f(x_0)}{h}$$

Why do these work?

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} \sum_{i=1}^{n} hf(a+ih) \quad Where: \quad h = \frac{b-a}{n}$$





Types of derivatives & integrals

- *Analytic* (aka *symbolic*): math ➜ math
  - The kind you learned in calculus class
  - Example: $\partial \sin(x) / \partial x$ ➜ $\cos(x)$
- *Numerical*: numbers ➜ numbers
  - Don't need explicit f(x), just some of its values
  - Example: $(y_1 - y_0) / (x_1 - x_0)$ ➜ $\Delta y/\Delta x$
- *Autodiff*: code ➜ code
  - Like analytic, but for large chunks of code
  - Aims for speed, not just correctness

All of these can be done with computers

Numerical derivatives & integrals come from approximation methods

- You have learned some methods for approximating a function f(x) (Examples?)
- We used functions like polynomials which have easy analytic derivatives & integrals
- We can also use these to estimate the derivative & integral of the true function f(x)

Recall: Lagrange polynomials

$$L_{n,j}(x) = \frac{(x-x_0)(x-x_1)\dots(x-x_{j-1})(x-x_{j+1})\dots(x-x_n)}{(x_j-x_0)(x_j-x_1)\dots(x_j-x_{j-1})(x_j-x_{j+1})\dots(x_j-x_n)}$$

Same

$$\boxed{L_{n,j}(x) \equiv \prod_{\substack{i=0\\ i\neq j}}^{n} \frac{(x-x_i)}{(x_j-x_i)}},$$

Basis

Lagrange

$$\boxed{P_n(x) \equiv \sum_{j=0}^{n} f(x_j) L_{n,j}(x)}$$

Data Basis

Lagrange error term

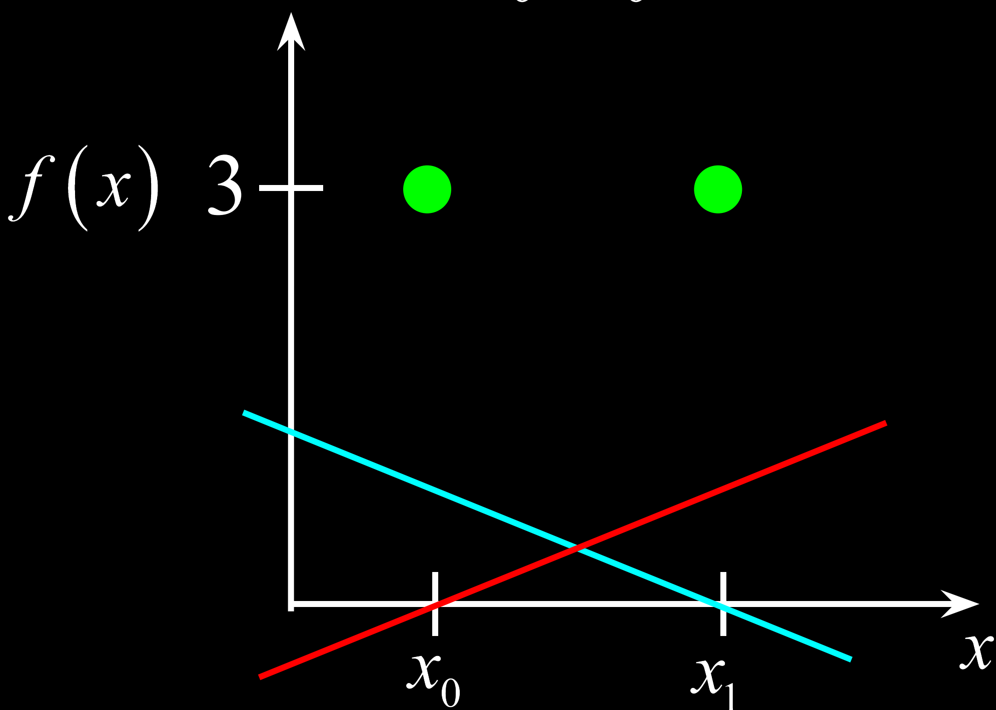
$$\boxed{f(x) = P_n(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\dots(x-x_n)}$$

Can approximate any function f(x) using data points at x_0, x_1, ... x_n. Sensitive to noise after 4 or 5 terms.

Lagrange polynomial for 2 points

Find the Lagrange polynomial for this data:

$$x_0, y_0 = (1, 3),\ x_1, y_1 = (2, 3)$$



$$L_0(x) = \frac{x - x_1}{x_0 - x_1}$$

$$L_1(x) = \frac{x - x_0}{x_1 - x_0}$$

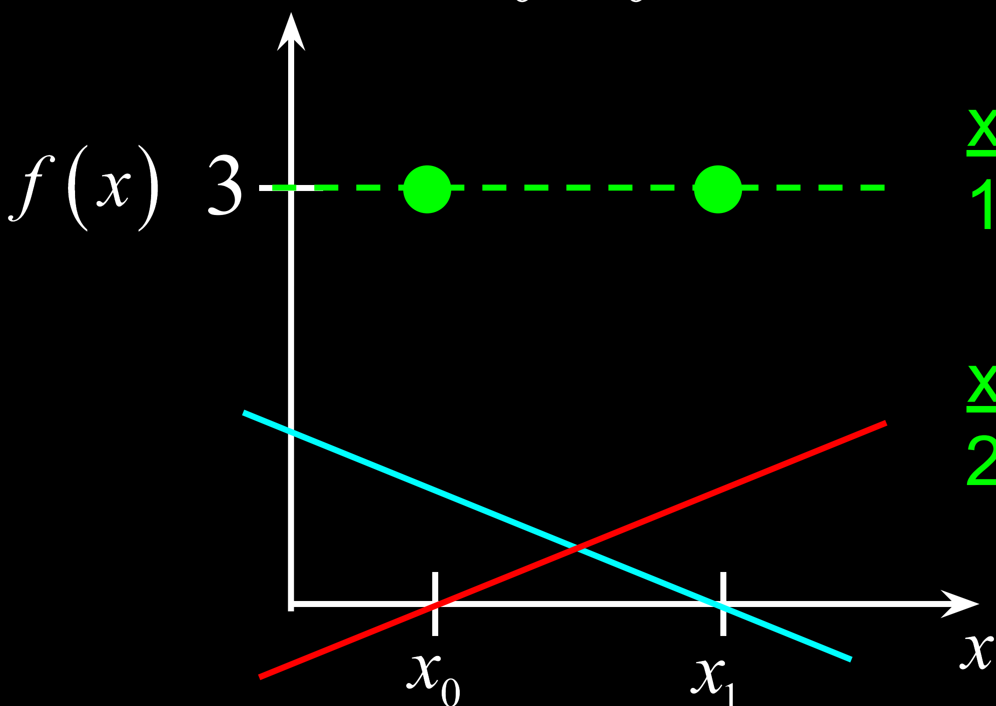
With a weighted sum of these two lines, you can make any line!

$$P(x) = L_0(x) f(x_0) + L_1(x) f(x_1) \approx f(x) \; on \left[x_0, x_1\right]$$

Lagrange polynomial for 2 points

Find the Lagrange polynomial for this data:

$$x_0, y_0 = (1, 3), x_1, y_1 = (2, 3)$$



$$\frac{x - 2}{1 - 2} \qquad \boxed{L_0(x) = \frac{x - x_1}{x_0 - x_1}}$$

$$\frac{x - 1}{2 - 1} \qquad \boxed{L_1(x) = \frac{x - x_0}{x_1 - x_0}}$$

In this case, a constant: 3

$$(2 - x) * 3 + (x - 1) * 3 = \mathbf{3}$$

$$\boxed{P(x) = L_0(x) f(x_0) + L_1(x) f(x_1) \approx f(x) \; on \; [x_0, x_1]}$$

Lagrange math & code

$$L_{n,j}(x) \equiv \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)},$$

$$P_n(x) \equiv \sum_{j=0}^{n} f(x_j) L_{n,j}(x)$$

```python
def lagrange_Lj(x, j, x_data):
    '''
    Returns the j-th Lagrange function L,
        as defined by the points x_data,
        evaluated at point x.
    '''
    L = 1.0
    for i in range(len(x_data)):
        if i==j:
            continue
        dx = x - x_data[i]
        x_interval = x_data[j] - x_data[i]
        L *= dx / x_interval
    return L
```

```python
def lagrange_P(x, x_data, y_data):
    '''
    Returns the value of the Lagrange
        polynomial P defined by x_data,
        y_data, evaluated at point x.
    '''
    assert len(x_data) == len(y_data)
    P = 0.0
    for j in range(len(x_data)):
        Lj = lagrange_Lj(x, j, x_data)
        P += y_data[j] * Lj
    return P
```

Lagrange for estimating df/dx

From p. 169 of F&B:

$$\boxed{f'(x_k) = \sum_{j=0}^{n} f(x_j) L'_{n,j}(x_k) + R_{n+1}(\xi(x_k))}$$

- We can use the derivative of the <u>Lagrange polynomial</u> for a set of N points $\{x_j, f(x_j)\}$ to approximate the derivative of the function f
- This is called the <u>N-point formula</u> for approximating the derivative of f
- For N points, polynomial order n = N-1 (Why?)

Lagrange polynomial derivatives

Use the Lagrange polynomials (from the definition), then find their derivatives:

$$L_{2,0}(x)=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} \rightarrow L'_{2,0}(x)=\frac{2x-x_1-x_2}{(x_0-x_1)(x_0-x_2)}$$

$$L_{2,1}(x)=\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} \rightarrow L'_{2,1}(x)=\frac{2x-x_0-x_2}{(x_1-x_0)(x_1-x_2)}$$

$$L_{2,2}(x)=\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} \rightarrow L'_{2,2}(x)=\frac{2x-x_0-x_1}{(x_2-x_0)(x_2-x_1)}$$

Lagrange polynomials for n=2 Lagrange derivatives for n=2

Example: 2-point derivative $f`(x_0)$

Assume we know $f(x_0)$ & $f(x_1)$, where $x_1 = x_0 + h$

$$f'(x_0) = \sum_{j=0}^{1} f(x_j) L'_{1,j}(x_0) + \boxed{\frac{f^{(1+1)}(\xi(x_0))}{(1+1)!} \prod_{\substack{j=0 \\ j\neq 0}}^{1} (x_0 - x_j)}$$

Error term

$$= \sum_{j=0}^{1} f(x_j) L'_{1,j}(x_0) + \boxed{\frac{f''(\xi)}{2}(x_0 - x_1),} \quad x_0 \le \xi \le x_0 + h$$

$$= f(x_0) L'_{1,0}(x_0) + f(x_1) L'_{1,1}(x_0) \boxed{- \frac{h}{2} f''(\xi),} \quad x_0 \le \xi \le x_0 + h$$

$$= \boxed{\frac{f(x_0 + h) - f(x_0)}{h} \boxed{- \frac{h}{2} f''(\xi)}}, \quad x_0 \le \xi \le x_0 + h$$

Error proportional to h

Example: 3-point derivative f`(x$_1$)

$$f'(x_k)=\sum_{j=0}^{2} f(x_j)L'_{2,j}(x_k)+\frac{f'''(\xi(x_k))}{3!}\prod_{\substack{j=0\\ j\neq k}}^{2}(x_k-x_j)$$

$$=f(x_0)\frac{2x_k-x_1-x_2}{(x_0-x_1)(x_0-x_2)}+f(x_1)\frac{2x_k-x_0-x_2}{(x_1-x_0)(x_1-x_2)}$$

$$+f(x_2)\frac{2x_k-x_0-x_1}{(x_2-x_0)(x_2-x_1)}+\frac{f'''(\xi(x_k))}{3!}\prod_{\substack{j=0\\ j\neq k}}^{2}(x_k-x_j)$$

$$\rightarrow \quad \boxed{f'(x_1)=\frac{f(x_1+h)-f(x_1-h)}{2h}-\frac{h^2}{6}f'''(\xi(x_1))}$$

Midpoint

f`(x$_1$)

Example: 3-point derivative f`(x_0)

$$f'(x_k)=\sum_{j=0}^{2} f(x_j)L'_{2,j}(x_k)+\frac{f'''(\xi(x_k))}{3!}\prod_{\substack{j=0\\ j\neq k}}^{2}(x_k-x_j)$$

$$=f(x_0)\frac{2x_k-x_1-x_2}{(x_0-x_1)(x_0-x_2)}+f(x_1)\frac{2x_k-x_0-x_2}{(x_1-x_0)(x_1-x_2)}$$

$$+f(x_2)\frac{2x_k-x_0-x_1}{(x_2-x_0)(x_2-x_1)}+\frac{f'''(\xi(x_k))}{3!}\prod_{\substack{j=0\\ j\neq k}}^{2}(x_k-x_j)$$

Midpoint f`(x_1)

$$\rightarrow \quad \boxed{f'(x_1)=\frac{f(x_1+h)-f(x_1-h)}{2h}-\frac{h^2}{6}f'''(\xi(x_1))}$$

Endpoint f`(x_0)

$$\rightarrow \quad \boxed{f'(x_0)=\frac{-3f(x_0)+4f(x_0+h)-f(x_0+2h)}{2h}+\frac{h^2}{3}f'''(\xi(x_0))}$$

Derivatives for N = 2, 3, & 5

$$2\ pt.:\quad f'(x_0)=\frac{f(x_0+h)-f(x_0)}{h}-\frac{h}{2}f''(\xi),\quad x_0\le\xi\le x_0+h$$

$$3\ pt.:\quad f'(x_0)=\frac{1}{2h}\left[-3f(x_0)+4f(x_0+h)-f(x_0+2h)\right]+\frac{h^2}{3}f'''(\xi),$$

$$(Endpoint)\quad x_0\le\xi\le x_0+2h$$

$$3\ pt.:\quad f'(x_0)=\frac{1}{2h}\left[f(x_0+h)-f(x_0-h)\right]-\frac{h^2}{6}f'''(\xi),$$

$$(Midpoint)\quad x_0-h\le\xi\le x_0+h$$

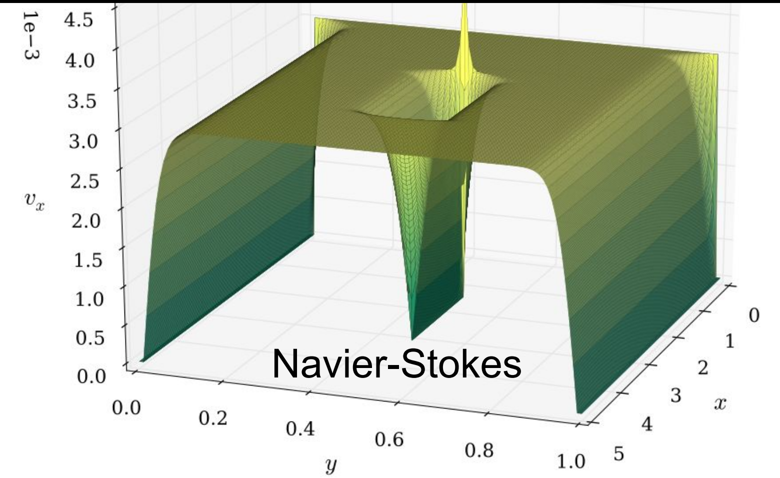
$$5\ pt.:\quad f'(x_0)=\frac{1}{12h}\left[f(x_0-2h)-8f(x_0-h)+8f(x_0+h)-f(x_0+2h)\right]$$

$$(Midpoint)\qquad -\frac{h^4}{30}f^{(5)}(\xi),\quad x_0-2h\le\xi\le x_0+2h$$

Which N is best?

Numerical differentiation summary

- N-point formula = Lagrange polynomial with different numbers of points (N = 2, 3, 5, etc.)
- Midpoint is better (less approx. error, fewer function evaluations, less round-off)
- Methods are unstable as $h \rightarrow 0$ (Why?)
  - Use $h > 10^{-8}$, and test (sensitivity analysis)
- We'll use these to solve BVPs / PDEs on a grid (Finite Difference Methods)



Numerical derivatives in Python

- np.gradient(y, x) estimates dy/dx using central difference (at the ends, endpoint difference)

```
def test_np_gradient():
    xx = np.linspace(-0.1, np.pi + 0.1, 100)
    yy = np.sin(xx)
    dyy = np.cos(xx)
    dyy_approx = np.gradient(yy, xx)
    error = dyy_approx - dyy
    mean_abs_error = np.mean(np.abs(error))
    print(f'{mean_abs_error = }')
```

- Result:

```
mean_abs_error = 0.00015
```

Analytic derivatives in Python

- Module **sympy** can give analytic gradients

```
def test_sympy():
    import sympy
    x, k = sympy.symbols('x k')
    f = sympy.sin(x**2 + k)
    df = sympy.diff(f, x)
    print(f'{f = }')
    print(f'{df = }')
```

- Result:

Try it!

Analytic derivatives in Python

- Module **sympy** can give analytic gradients

```
def test_sympy():
    import sympy
    x, k = sympy.symbols('x k')
    f = sympy.sin(x**2 + k)
    df = sympy.diff(f, x)
    print(f'{f = }')
    print(f'{df = }')
```

- Result:

```
f = sin(k + x**2)
df = 2*x*cos(k + x**2)
```

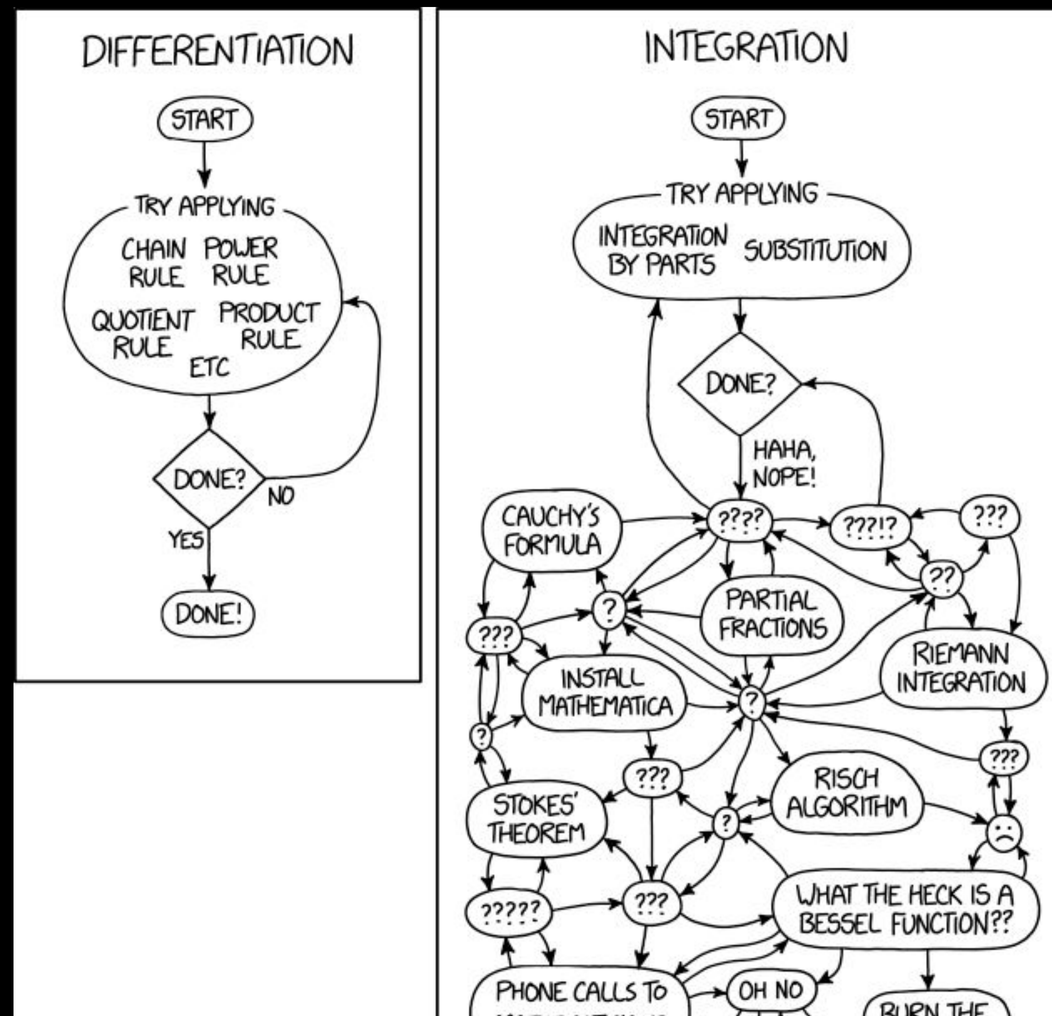
Autodiff in Python

- Frameworks like PyTorch, TensorFlow, and Jax can autodiff a whole program
  - Unlike an analytic derivative, autodiff is not limited to single mathematical expressions
  - Just code your function f(), even with loops, then request the derivative
- At Schrodinger, here's how we get atomic forces from our potential energy models:

```
forces = -torch.autograd.grad(energy, xyz)
```

Derivatives vs integrals

- Many families of functions are "*closed*" under differentiation but not integration
  - This is why integrals stink
- Fortunately, we have numerical integration



xkcd.com/2117

Numerical integration

- Numerical integration (aka quadrature) refers to a method which approximates an integral using a weighted sum of function values:

$$\int_a^b f(x)\,dx \approx \sum_{j=0}^{n} \alpha_j f(x_j)$$

- We can pick any set of n+1 points $x_0 \ldots x_n$ in [a,b] we would like, but to start we'll assume we have equally spaced ones
- What are the "weights"?

Quadrature for small n

- We would like to use many points (large n), since that makes h (the <u>interval size</u>) small:

$$\int_a^b f(x)\,dx = \lim_{n\to\infty}\sum_{i=1}^{n} hf(a+ih) \approx \sum_{j=0}^{n}\alpha_j f(x_j)$$

$$Where: \quad h = \frac{b-a}{n}$$

- We don't always have a lot of data – sometimes only two or three points
- We can use Lagrange polynomials (just like for differentiation) to derive formulae. . .

Simple Quadrature Rules

Midpoint rule (Figure 4.1 on F&B p108):

$$\int_a^b f(x)\,dx = (b-a)\,f\left(\frac{a+b}{2}\right) + \frac{f''(\xi)}{24}(b-a)^3 \approx \sum_{j=0}^{0} \alpha_j f(x_j)$$

$$= \boxed{2hf(a+h) + \frac{h^3}{3} f''(\xi) \quad for\ h = \frac{b-a}{2}}$$

Which is more accurate?

Trapezoidal rule (Fig 4.2, F&B p110):

$$\int_a^b f(x)\,dx = (b-a)\frac{f(a)+f(b)}{2} - \frac{f''(\xi)}{12}(b-a)^3 \approx \sum_{j=0}^{1} \alpha_j f(x_j)$$

$$= \boxed{h\left[f(a)+f(b)\right] - \frac{2h^3}{3} f''(\xi) \quad for\ h = \frac{b-a}{2}}$$

Simpson's Rule (n = 2)

- The midpoint and trapezoid rules are fine, error $O(h^3)$, but Simpson's Rule is $O(h^5)$:

$$\int_a^b f(x)\,dx \approx \sum_{j=0}^{2} \alpha_j f(x_j)$$

$$= \frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] - \frac{f^{(4)}(\xi)}{2880}(b-a)^5$$

$$= \boxed{\frac{h}{3}\left[f(a) + 4f(a+h) + f(a+2h)\right] - \frac{h^5}{90} f^{(4)}(\xi)} \left(h = \frac{b-a}{2}\right)$$

- What if the interval [a,b] is very large?

Composite Quadrature

- If we want to integrate over a big interval [a,b], we can break it up into smaller parts and do basic quadrature on each part:

$$\int_a^b f(x)\,dx = \int_a^{a+h} f(x)\,dx + \int_{a+h}^{a+2h} f(x)\,dx + \ldots + \int_{b-h}^{b} f(x)\,dx$$

- This method is called <u>Composite Quadrature</u> and the resulting rules are at F&B p118-119
- Makes h smaller, so error is much smaller
- Only works if we can get more values of f(x)

How do we pick h?

- Adaptive Quadrature uses a bound on the approximation error ε to choose the number of subintervals – example in F&B uses Simpson's Rule on four subintervals (p. 140)
- Gaussian Quadrature minimizes the error of approximation by picking exactly the right points (given the approximating polynomial), producing a variable step size h

More complicated situations?

- Multiple integrals (often double and triple): useful for simulators (CAD programs, fluid dynamics) where you need to calculate properties over complex 3D shapes

$$\iiint_{\Omega} f(x)\, dx_1 dx_2 dx_3$$

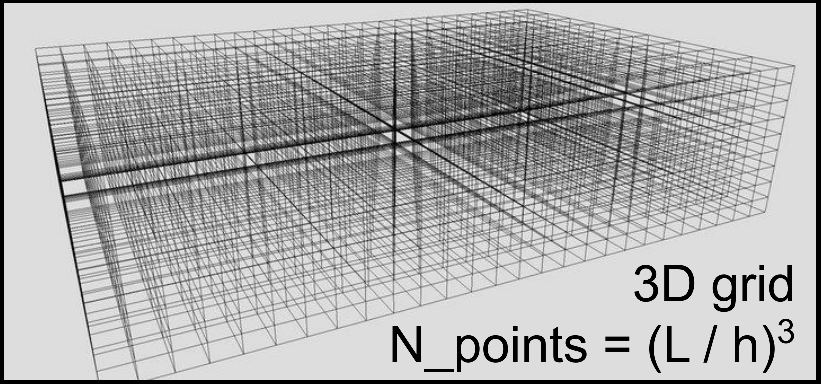
- Improper integrals (some bounds ∞): hope integral converges fast enough that it can be estimated with large but finite bounds

Quadrature functions

- In Python, scipy.integrate.quad(f, a, b) finds the integral of f on [a,b] using adaptive quadrature with a specified error tolerance
- dblquad and tplquad in scipy.integrate will do double and triple integrals (much slower)
- What about higher dimensions?

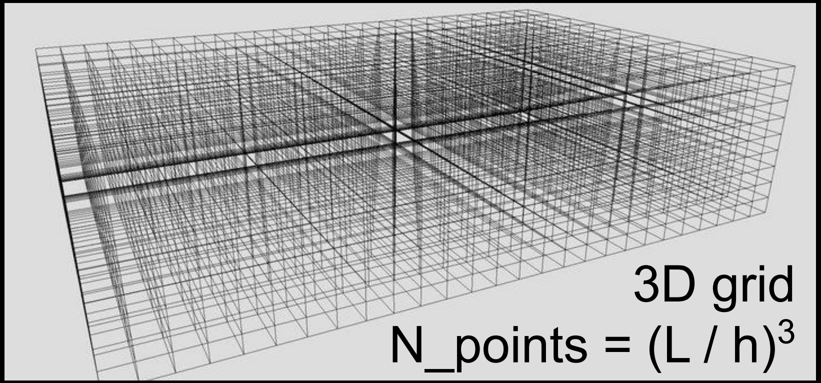
Curse of exponentiality

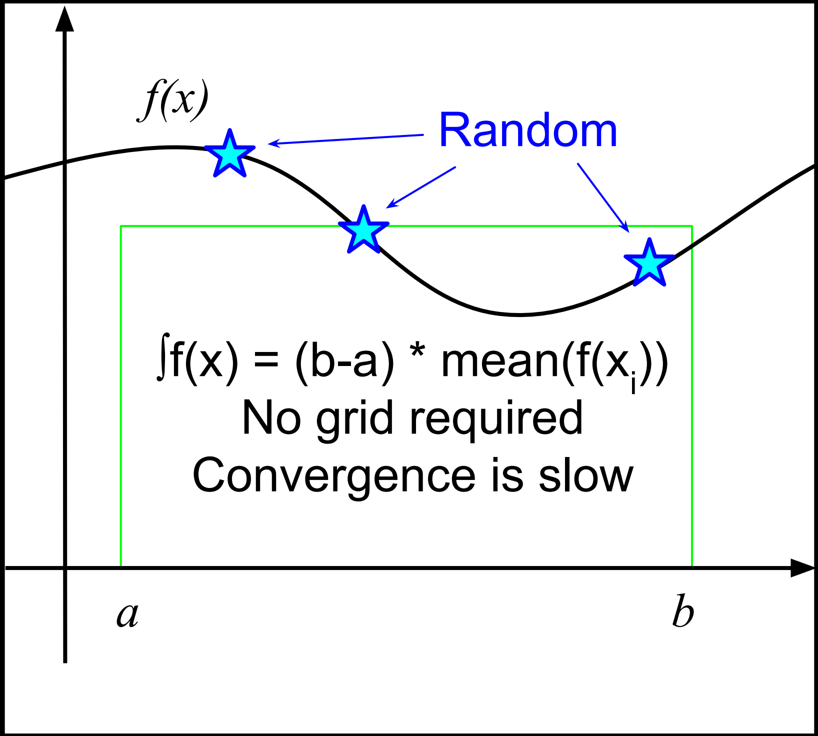
- Quadrature fails for high-dimensional grids
- Grid size grows exponentially



Monte Carlo integration

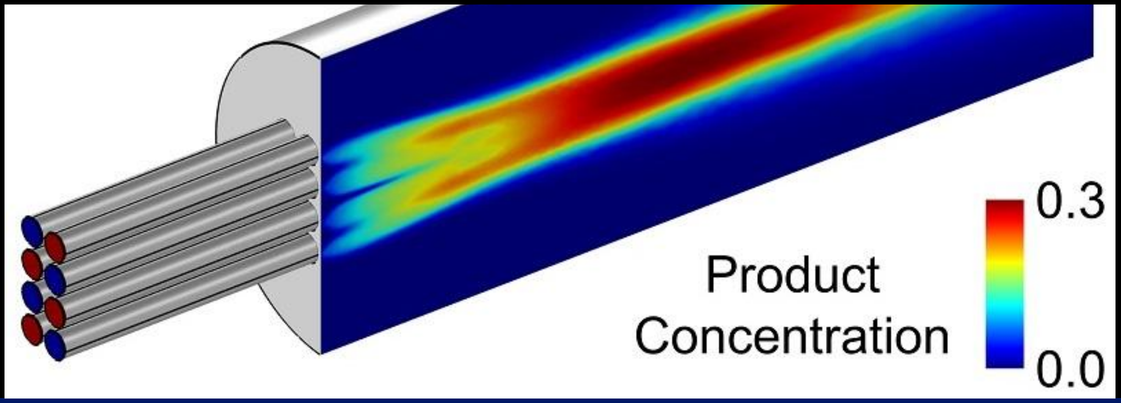
- Quadrature fails for high-dimensional grids
- Grid size grows exponentially
- Monte Carlo: pick points *at random* & compute mean(f)
- Standard deviation gives uncertainty estimate for mean(f)





Monte Carlo example: reactor

- You're estimating reactor output given yield $f(x_i)$ for concentrations x_i of reactants & impurities
- For each impurity, you have <u>bounds</u> on the concentration, not exact amounts
- Solution: generate random points within the bounds, calculate expected yield mean($f(x_i)$), multiply by total input to get the total output

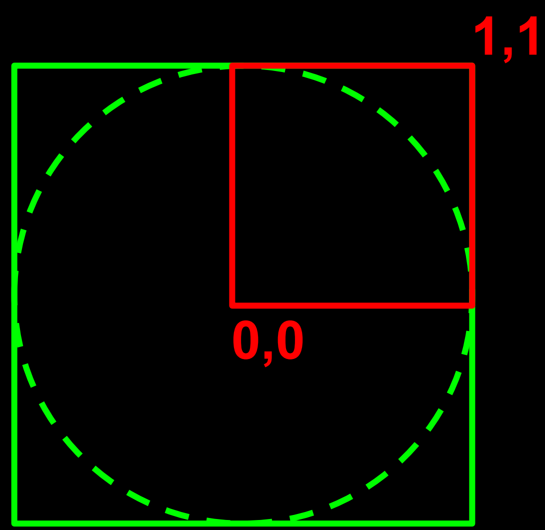


How do we get random numbers?

- For numerical methods, we don't try to get real random numbers (such as quantum noise)
- We use functions that give <u>repeatable</u> outputs with the <u>same statistical properties</u> as real random numbers
- These functions are *Pseudo-Random Number Generators* (PRNGs)
- Found in **np.random** & **hashlib**

Monte Carlo example: π

- Say we have a function f(x, y) = 1 when the point x, y is in the unit circle, otherwise 0
- Find the average value of this function in the square from 0,0 to 1,1 (see np.random.random)



- How is this value related to π?
- How many random points does it take to converge π reliably to 3.14?

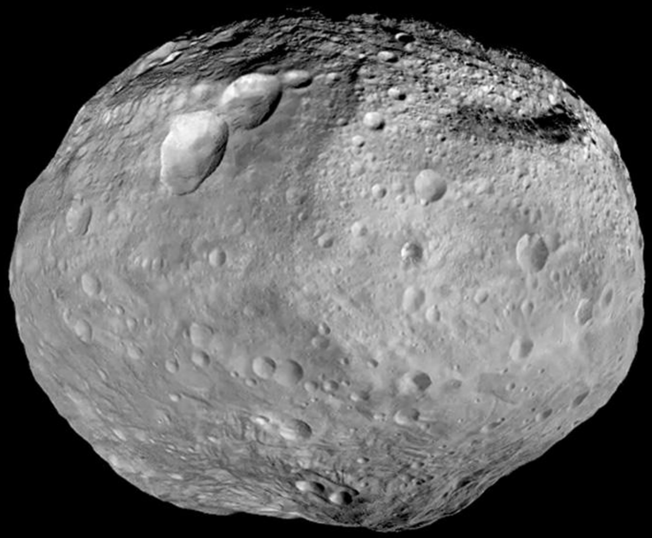
Integrating motion: Asteroids

- This is how I got into numerical methods
- Sitting in Cooper Union physics class, not paying much attention, coding video games where things move around (things like asteroids)

Integrating motion: Asteroids

Given an asteroid with a position, velocity, and forces, how does it move?

Position = x
Velocity = dx/dt
Force = $m(d^2x/dt^2)$



If we know the ***initial values*** of the variables, we can find their values at later times too using a form of numerical integration

Integrating motion: instability

Unlike quadrature, Initial Value Problems (IVPs) can suffer from *instability*

Integrating motion: instability

All reading for next week: Intro to Initial Value Problems, Euler's method, RK4: PNM 22.1-5. More details in F&B 5.1-3.