Numerical integration is done by fitting a polyomial and finding a weighted-sum. Numerical differentiation is done by brute-forcing the limit.

---

numerical-integration:~# Numerical integration, derived from polynomial interpolations, is a *weighted sum* which can be programmed iteratively. Newton-Cotes integration uses rational weights for evenly spaced data while Gaussian quadrature uses Legendre roots weights for unevenly spaced data. A rule of thumb is an inverse relation between approximation accuracy and noisiness of data guiding our choice for the appropriate technique_

```
…
s = 0.5*f(a) + 0.5*f(b)
for k in range(1,N):
    s += f(a+k*h)
    return h*s
```

First-order Newton-Cotes integration is called trapezoidal rule. Observe that it has 0.5 weights at boundaries a and b and unity weights at the middle.

```
def gaussxwab(N,a,b):
    x,w = gaussxw(N)
    return 0.5*(b-a)*x+0.5*(b+a),0.5*(b-a)*w
```

Gaussain quadrature weights can be directly called from a python package by assigning weights at domain [-1,1] and rescaling

! Pitfall: Mind the order parity of Newton-Cotes method. Plugging in odd slices for, say, Simpson's rule will yield an oscillating error.

approximation-error:~# A closed form expression for the approximation error for each iteration of an integration scheme allows us to precisely "budget" computational power-to-error ratio. Extensive exploitation of this leads to *Romberg integration*: an add-on to Newton-cotes integration for more accuracy_

```
…
s = 0.5*f(a) + 0.5*f(b)
for k in range(1,N):
    s += f(a+k*h)
    error = abs((1/3)*(((1/2)*I+(h*t))-I))
    I = ((1/2)*I+(h*t))
```

Information about error at each iteration coded within the loop can be used to halt the loop as needed.

```
T[i][m] = T[i][m-1] +
((1)/((4**m)-1))*(T[i][m-1]-T[i-1][m-1])
```

Data about the 2D romberg integration can be stored in an array and calling the latest version as needed

! Pitfall: Make sure to use the latest version (max. i and m) for Romberg integration. Error oscillates periodically throughout the Romberg cycle.

numerical-derivative:~# A numerical method to derivatives is a brute-force approach to the the limit definition. The central difference method offers the best accuracy compared to the one-sided forward and backward difference methods_

```
    def d_f(x):
      return ((f(x+(h/2))-f(x-(h/2)))/h)
```

Numerical derivatives are straightforward to code by approaching h -> 0 through brute force than analytical limit evaluation.

! Pitfall: Especially with derivatives, watch out for noisy data. We can smoothen them via Fourier transforms or interpolate a polynomial.