Numerical Integration

Charlie Murry

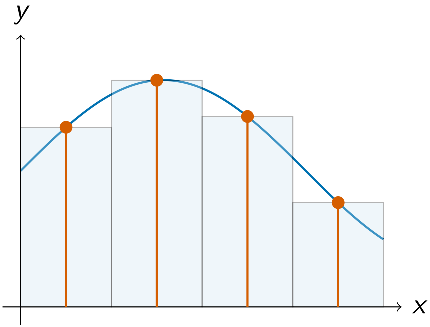
January 2025

Numerical Integration

- General form: $\int_I f(x)w(x)\,dx \approx \sum_{i=0}^{n} w_i f(x_i)$
- Components:
  - Nodes (x_i)
  - Weights (w_i)

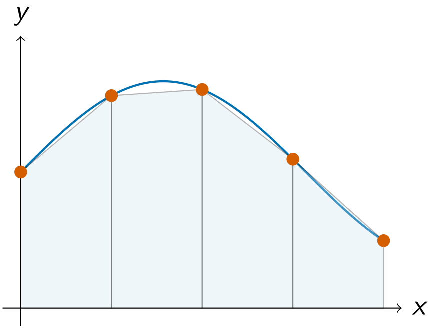
Midpoint Rule

- Approximates integral using rectangles
- Node at midpoint of each interval
- Simple but less accurate than other methods



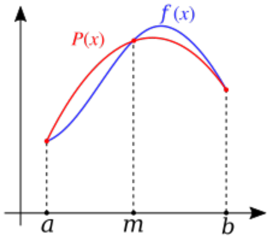
Trapezoidal Rule

- Uses linear approximation between points
- Formula: $\int_a^b f(x)dx \approx \frac{b-a}{2n}[f(x_0) + 2f(x_1) + ... + f(x_{n+1})]$
- More accurate than midpoint rule



Simpson's Rule

- Uses parabolic approximation
- Formula: $\int_a^b f(x)dx = \frac{h}{3}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$
- where $h = \frac{b-a}{n}$ and $n = 2$ step size (three points).
- More generally: nodes are evenly spaced: $x_i = a + ih$
- Weights:
  - $w_0 = w_n = \frac{h}{3}$
  - $w_i = \frac{4h}{3}$ for even i
  - $w_i = \frac{2h}{3}$ for odd i



Monte Carlo Integration

- Tecnique using (psuedo-)randomly drawn nodes, equally weighted.
- Relies on the Strong Law of Large Numbers (SLLN).

  Sample average converges almost surely to the expected value.

$$\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} f(x_i) = E[f(\tilde{X})]$$

- Nodes: $x_i \rightarrow$ draw from uniform on the computer.
- Weights: $w_i = \frac{1}{n}$

Simple Julia Script

```
using Statistics

function mc_integral(f, a, b, n)
    # Generate random points in interval [a,b]
    x = a .+ (b-a) * rand(n)

    # Compute function values and scale by interval width
    y = f.(x) * (b-a)

    # Return mean and standard error
    return mean(y), std(y)/sqrt(n)
end

# Test with f(x) = x^2 on [0,1]
f(x) = x^2
n = 100000
result, error = mc_integral(f, 0, 1, n)

println("Integral of x^2 from 0 to 1:")
println("Monte Carlo: $result ± $error")
println("Exact: $(1/3)")
```

Quasi-Monte Carlo Methods

- Having truley random numbers may waste guesses.
- We can choose nodes that are "low-discrepency"
- Improvements over standard Monte Carlo:
  - Halton/Sobol sequences for better distribution.
  - Antithetic acceleration: if using x_1, also use $-x_1$.
- More systematic coverage of integration domain.

Gaussian Quadrature

- Nodes and weights chosen "efficiently."
- Fewwst nodes possible to achieve exact approximation for polynomials.
- Even though you may not be dealing with a polynomial, this type of result can be useful in having confidence in your result.
- Choose x_i and w_i so that aprox is exact with n nodes if f is P_{2n-1}
- Example: Match $2n$ moments of weight function $g(x)$

Gaussian Quadrature Example

Example: Match $2n$ moments of weight function $g(x)$
Systems of $2n$ equations:

$$\int_a^b x^k g(x) dx = \sum_{i=1}^{n} w_i x_i^k, \quad k = 0...2n-1$$

If $g(x)$ is a density, then the k equations are the k uncentered moments of a cont. random variable.

If $x \sim N(0,1)$ and $\phi(x)$ is the pdf (weights)

Then the first six moments are

$$E(X^0) = 1 = \sum_{i=1}^{3} w_i$$

$$E(X^1) = 0 = \sum_{i=1}^{3} w_i x_i$$

$$E(X^5) = 0 = \sum_{i=1}^{3} w_i x_i^5$$

Solve this system of six equations and six unknowns.

x_i	w_i
-1.732	0.166
0	0.667
1.732	0.166

If you want to find

$$Var(X) = E(X^2) = \int x^2 \phi(x) dx$$

Then now we know.

$$\int x^2 \phi(x) dx = \sum_{i=1}^{3} w_i x_i = (-1.732)^2(0.166) + (0)^2(0.667) + (1.732)^2(0.166)$$

What is the big deal?

- For certain weight functions, people have worked out exactly which nodes and wights you should use if you want n nodes.
- In the case above, we were approximating the integrals of polynomials so we could get it exact.
- In your work, you probably won't have polynomials, but you can still use these x, n to get a high level of precision.