CS 450: Numerical Anlaysis[1]

Numerical Integration and Differentiation

University of Illinois at Urbana-Champaign

[1] *These slides have been drafted by Edgar Solomonik as lecture templates and supplementary material for the book "Scientific Computing: An Introductory Survey" by Michael T. Heath (slides).*

Integrability and Sensitivity

- Seek to compute $\mathcal{I}(f) = \int_a^b f(x)dx$:

- The condition number of integration is bounded by the distance $b - a$:

Quadrature Rules

- Approximate the integral $\mathcal{I}(f)$ by a weighted sum of function values:

$$\mathcal{I}(f) = \int_a^b f(x)\,dx \approx Q_n = \sum_{i=1}^{n} w_i f(x_i) = \langle w, y \rangle$$

Q_n: n-point; $x_1, \ldots, x_n$: nodes; y_i

- For a fixed set of n nodes, polynomial interpolation followed by integration give $(n-1)$-*degree quadrature rule*:

$$\mathcal{I}(f) \approx Q_n = \mathcal{I}(p) = \mathcal{I}\left(\left\langle \{e_i\}_{i=1}^n, c \right\rangle\right) = \left\langle \underbrace{V^{-T}\{\mathcal{I}(e_i)\}_{i=1}^n}_{w}, y \right\rangle$$

$V^{-1} y = \langle \{\mathcal{I}(e_i)\}_{i=1}^n, V^{-1} y \rangle$

p: degree $n-1$ polynomial interpolant of $(x_1, y_1), \ldots, (x_n, y_n)$

$$p(x) = \sum_{i=1}^{n} c_i e_i(x) \Rightarrow \int_a^b p(x)\,dx = \sum_{i=1}^{n} c_i \int_a^b e_i(x)\,dx$$

Determining Weights in a General Basis

- A quadrature rule provides x and w so as to approximate

$Q_n = \langle w, y \rangle$

- eqn. spaced
- Chebyshev points
- find optimal distribution of points, to maximize degree

- *Method of undetermined coefficients* obtains y from *moment equations* based on Vandermonde system:

weights w are independent of f or y and so are valid provided interval of integration is the same

Newton-Cotes Quadrature

Demo: *Newton-Cotes weight finder*

- *Newton-Cotes* quadrature rules are defined by equispaced nodes on $[a, b]$:

  can be either open or closed

  ↑ contains a, b

  correct construct

- The *midpoint rule* is the $n = 1$ open Newton-Cotes rule:

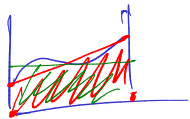
  $$M(f) = (b-a) f(m)$$

  $m = \frac{a+b}{2}$

  0th degree

  1st degree

- The *trapezoid rule* is the $n = 2$ closed Newton-Cotes rule:

  $$T(f) = \frac{b-a}{2}(f(b) + f(a))$$

  1st degree

- *Simpson's rule* is the $n = 3$ closed Newton-Cotes rule:

  quadratic approximation (interpolant)

  2nd degree

Error in Newton-Cotes Quadrature

Demo: *Accuracy of Newton-Cotes*

- Consider the Taylor expansion of f about the midpoint of the integration interval $m = (a+b)/2$:

$$f(x) = f(m) + f'(m)(x-m) + \frac{f''(m)}{2}(x-m)^2 + \dots$$

Integrating the Taylor approximation of f, we note that the odd terms drop:

$$I(f) = (b-a)f(m) + \frac{f''(m)}{24}(b-a)^3 + O((b-a)^5)$$

$M(f)$ $\qquad$ $E(f)$

midpoint rule is third order in accuracy

→ to bound error in Trapezoid rule want $f(m)$ in terms of $f(a)$, $f(b)$

Error Estimation

- The trapezoid rule is also first degree, despite using higher-degree polynomial interpolant approximation, since

$$f(m) = \frac{1}{2}\left[f(a) - f'(m)\left(\frac{b-a}{2}\right) - \frac{f''(m)}{2}\left(\frac{b-a}{2}\right)^2 - \dots \right.$$

$$\left. + f(b) - f'(m)\left(\frac{b-a}{2}\right) - \frac{f''(m)}{2}\left(\frac{b-a}{2}\right)^2 - \dots \right]$$

- The above derivation allows us to obtain an error approximation via a difference of midpoint and trapezoidal rules:

$$I(f) = (b-a)\left[\frac{1}{2}(f(b) + f(a))\right] - \frac{f''(m)}{12}(b-a)^2 - O\left(\left(\frac{b-a}{2}\right)^5\right)$$

$T(f)$

$2E(f)$

$$3E(f) = M(f) - T(f) + O\left((b-a)^5\right)$$

Error in Polynomial Quadrature Rules

- We can bound the error for a an arbitrary polynomial quadrature rule by

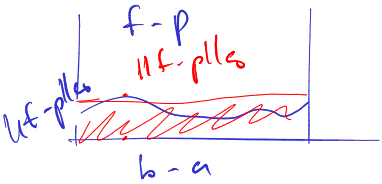
$$|I(f) - Q_n(f)| = |I(f) - I(p)| \quad \text{(polynomial interpolant)}$$

$$= |I(f-p)|$$

$$\leq (b-a)\,\|f-p\|_\infty$$

$$\leq (b-a)\,(b-a)^n\,|f^{(n)}|_\infty$$

$$= O((b-a)^{n+1})$$



Conditioning of Newton-Cotes Quadrature

- We can ascertain stability of quadrature rules, by considering the amplification of a perturbation $\hat{f} = f + \delta f$:

$\hat{y}_i = \hat{f}(x_i)$

$y_i = f(x_i)$

$$Q_n(\hat{f}) - Q_n(f) = \langle w, \hat{y} \rangle - \langle w, y \rangle$$

$$= \langle w, \hat{y} - y \rangle$$

$$= Q_n(\delta f) \leq \|w\|_1 \, \|\delta f\|_{\max}$$

condition: $\sum_{i=1}^{n} w_i = b - a$

want to maximize $\|w\|_1 = \sum_{i=1}^{n} |w_i| \geq b - a$

- Newton-Cotes quadrature rules have at least one negative weight for any $n \geq 11$:

bad

So Newton-Cotes quadrature rules can be ill-conditioned (suboptimal) although for $n = 1, 2, 3$ they are fine

Clenshaw-Curtis Quadrature

- To obtain a more stable quadrature rule, we need to ensure the integrated interpolant is well-behaved as n increases:

Chebyshev-spaced nodes

↳ interpolation error is smaller

↳ can show that $w_i > 0$

Gaussian Quadrature

Demo: *Gaussian quadrature weight finder*

- So far, we have only considered quadrature rules based on a fixed set of nodes, but we may also be able to choose nodes to maximize accuracy:

exactly n parameters, for a total of (weights & nodes) $2n$ degrees of freedom

degree $2n-1$ quadrature rules

picks x, w

- The *unique* n-point *Gaussian quadrature rule* is defined by the solution of the nonlinear form of the moment equations in terms of *both* $\boldsymbol{x}$ and $\boldsymbol{w}$:

solve for x and w $\quad V(x, \{\phi_i\}_{i=1}^n)^T w = \{I(\phi_i)\}_{i=1}^n$

independent of $y(f)$, dependent only on n, a, b

- solution generally exists and is unique
- $w_i > 0$, well-conditioned

Using Gaussian Quadrature Rules

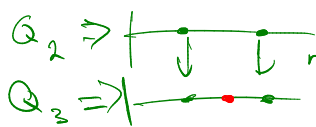
- Gaussian quadrature rules are hard to compute, but can be enumerated for a fixed interval, e.g. $a = 0, b = 1$, so it suffices to transform the integral to $[0, 1]$

$$I(f) = \int_a^b f(x)\,dx = \frac{1}{b-a}\int_0^1 g(t)\,dt \qquad t = \frac{x-a}{b-a} \qquad dt = \frac{1}{b-a} \qquad g(t) = f(t(b-a)+a)$$

can be similar to get to $[-1, 1]$

- Gaussian quadrature rules are accurate and stable but not progressive (nodes cannot be reused to obtain higher-degree approximation):

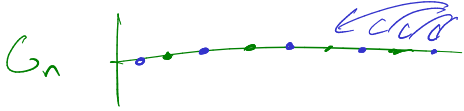
progressive rules reuse nodes, e.g.



require only one new function evaluation

Progressive Gaussian-like Quadrature Rules

- *Kronod* quadrature rules construct $(2n+1)$-point $(3n+1)$-degree quadrature K_{2n+1} that is progressive with respect to Gaussian quadrature rule G_n:



Kronod given G_n computing K_{2n+1} requires $n+1$ func. eval.

computing G_{2n+1} would require $2n+1$ degree $4n+1$

- *Patterson* quadrature rules use $2n+2$ more points to extend $(2n+1)$-point Kronod rule to degree $6n+4$, while reusing all $2n+1$ points.
- Gaussian quadrature rules are in general open, but *Gauss-Radau* and *Gauss-Lobatto* rules permit including end-points:

Gauss rules are open
Gauss-Lobatto is closed | Gauss-Radau contains 1-endpoint

Composite and Adaptive Quadrature

- *Composite quadrature rules* are obtained by integrating a piecewise interpolant of f:

polynomial

main advantage is accuracy $\Rightarrow$ shrink $b-a$ (interval of integration)

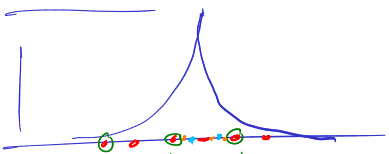
$$I(f) \approx I(p) = \sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} p_i(x)\,dx$$

$x_0 = a$, $x_n = b$

p_i: ith piece

$E(f) \approx O((b-a)^{\wedge})$ ← error

- Composite quadrature can be done with adaptive refinement:

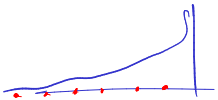


e.g. $E(f) \approx \frac{1}{3}(4I(f) - T(f))$

error near quickly changing part of function converges slower (estimate says it's large)

More Complicated Integration Problems

- To handle improper integrals can either transform integral to get rid of infinite limit or use appropriate open quadrature rules.



- Double integrals can simply be computed by successive 1-D integration.

if can decouple

- High-dimensional integration is often effectively done by *Monte Carlo*:

integration volume ← Ω ; N ← number of samples N

$$I(f) \simeq \frac{\Omega}{N} \sum_{i=1}^{N} f(x_i)$$

x_i are sample from domain

error converges with $O(1/\sqrt{N})$

Integral Equations

- Rather than evaluating an integral, in solving an *integral equation* we seek to compute the integrand. A typical linear integral equation has the form

$$\int_a^b K(s,t)u(t)dt = f(s), \quad \text{where} \quad K \quad \text{and} \quad f \quad \text{are known.}$$

unknown

kernel known function values known

linear system matrix

$$\int_a^b k(s,t)\,u(t)\,dt \approx \sum_{j=1}^{n} w_j\, k(s_i,t_j)\,u(t_j)\,dt = f(s_i)$$

- Using a quadrature rule with weights $w_1, \ldots, w_n$ and nodes $t_1, \ldots, t_n$ obtain

Numerical Differentiation

Demo: Taking Derivatives with Vandermonde Matrices

- Automatic (symbolic) differentiation is a surprisingly viable option:

- Numerical differentiation can be done by interpolation or finite differencing:
  - Given polynomial interpolant, its derivative is easy to obtain by differentiating the basis in which it is expressed,

Accuracy of Finite Differences

Demo: *Finite Differences vs Noise*
Demo: *Floating point vs Finite Differences*

- *Forward and backward differencing* provide first-order accuracy:

- *Centered differencing* provides second-order accuracy:

Demo: Richardson with Finite Differences

Extrapolation Techniques

- Given a series of approximate solutions produced by an iterative procedure, a more accurate approximation may be obtained by *extrapolating* this series.

- In particular, given two guesses, *Richardson extrapolation* eliminates the leading order error term:

High-Order Extrapolation

- Given a series of k approximations, *Romberg integration* applies $(k-1)$-levels of Richardson extrapolation.

- Extrapolation can be used within an iterative procedure at each step: