CS 450: Numerical Anlaysis

Lecture 21

Chapter 7 Numerical Integration and Differentiation

Gaussian Quadrature, Integral Equations, and Numerical Differentiation

Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

April 6, 2018

Quadrature Rules

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

values

$$I(f) \approx Q_n(f) = \langle \boldsymbol{w}, \boldsymbol{y} \rangle, \quad \text{where} \quad y_i = f(x_i)$$

weights

$$I(f) = \int_a^b f(x)\,dx$$

$$Q_n(f) = I(p_{n-1}) = \overbrace{\left[I(\varphi_1(x)) \cdots I(\varphi_n(x)) \right]}^{s^T} V^{-1}(x, \{\varphi_i\}_{i=1}^n)\, y$$

$(n-1)$-degree interpolant of f

$$\int_a^b p_n(x)\,dx = \sum_{i=1}^n \underbrace{\int_a^b \varphi_i(x)\,dx}_{s_i}$$

$$I(p_{n-1}) = s^T V^{-1} y = \langle w, y \rangle$$

$$s^T V^{-1} = w^T$$

$$s^T = w^T V \Rightarrow \boxed{V^T w = s}$$

Gaussian Quadrature

- So far, we have only considered quadrature rules based on a fixed set of nodes, but we can also choose a set of nodes to improve accuracy:

Estimate $\mathcal{I}(f)$ by optimal choice of $\underbrace{\text{nodes}}_{n \text{ DoF}}$ and $\underbrace{\text{weights}}_{n \text{ DoF}}$. Gaussian quadrature rules use n points to obtain degree $2n-1$.

- 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 $\boldsymbol{w}$:

$$V(x, \{\phi_i\}_{i=1}^{2n-1}) = \begin{bmatrix} \phi_1(x_1) & \cdots & \phi_{2n-1}(x_1) \\ \vdots & & \vdots \\ \phi_1(x_n) & \cdots & \phi_{2n-1}(x_n) \end{bmatrix}$$

$V(x, \{\phi_i\}_{i=1}^{2n-1})^T s = w$ for a given x is overdetermined

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 $[0, 1]$

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

$$f(x) =: g\left(\frac{x+a-b}{b-a}\right)$$

can apply G_n - Gaussian Quadrature with n-points on $[0,1]$ for g gives $I(f)$

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

· maximal interpolation degree

· $W > 0$ for any G_n

so G_n and G_m for $n \neq m$ have no nodes in common

Progressive Gaussian-like Quadrature Rules

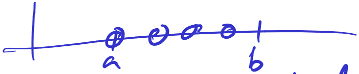
- *Kronod* quadrature rules construct $(2n+1)$-point quadrature K_{2n+1} that is progressive w.r.t. Gaussian quadrature rule G_n

  K_{2n+1} to have the n nodes of G_n while maximizing degree, which is $3n+1$.

  G_{2n+1} has degree $2(2n+1) - 1 = 4n+1$

- Gaussian quadrature rules are in general open, but Gauss-Radau and Gauss-Lobatto rules permit including end-points:

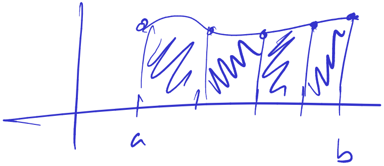
  Gauss-Radau include 1 end-point



  Gauss-Lobatto include a and b

Composite and Adaptive Quadrature

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



Composite midpoint rule

$$I(f) = \sum_{i=1}^{n-1} \underbrace{\int_{x_i}^{x_{i+1}} f(x)\,dx}_{(x_{i+1}-x_i)\, f\left(\frac{x_{i+1}+x_i}{2}\right)}$$

- Composite quadrature can be done with adaptive refinement:

high error!

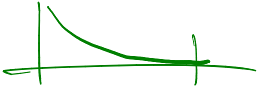
do refinement

we can estimate error, e.g. compare midpoint and Trapezoidal.

Adaptive mesh refinement

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.

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2} f(x,y)\,dx\,dy, \qquad \int_{\Omega} f(x,y)\,dx\,dy$$

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

$$\int_{\Omega} f(\vec{x})\,d\vec{x} \approx \frac{1}{N}|\Omega| \sum_{i=1}^{N} Y_i, \quad Y_i = f(x_i) \text{ for } x_i \text{ random in } \Omega$$

$$= E[\quad], \quad \text{error} \downarrow O\left(\frac{1}{\sqrt{N}}\right)$$

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

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

given given

$$\int_a^b K(s,t)\,u(t)\,dt \approx \sum_{i=1}^{n} K(s,t_i)\,u(t_i)\,w_i$$

$$\sum_{i=1}^{n} K(s_j,t_i)\,u(t_i)\,w_i = f(s_j) \quad A^T w = y$$

$$a_{ij} = w_i K(s_j,t_i), \quad y_j\ f(s_j)$$

- Integral equations are used to
  - recover signal u given response function with kernel K and measurements of f,
  - solve equations arising from Green's function methods for PDEs.

Challenges in Solving Integral Equations

- Integral equations based on response functions tend to be ill-conditioned, which is resolved using
  - truncated singular value decomposition of $\boldsymbol{A}$, where $a_{ij} = w_j K(s_i, t_j)$

$$Aw = y \qquad A = [\ ][\cdots][\ ]$$

ignore small singular values of A

  - replacing the linear system with a regularized linear least squares problem,

$$\min_w \|Aw - y\| + \lambda\|w\| \Rightarrow \min_w \left\| \begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix} w - \begin{bmatrix} y \\ 0 \end{bmatrix} \right\|$$

  - expressing the solution using a basis

$$u(t) = \sum_{i=1}^{m} c_i \phi_i(t)$$

Numerical Differentiation

- Automatic (symbolic) differentation is a surprisingly viable option.
  - Any computer program is differentiable, since it is an assembly of basic arithmetic operations.
  - Existing software packages can automatically differentiate whole programs.
- Numerical differentation can be done by interpolation or finite differencing
  - Given polynomial interpolant, its derivative is easy to obtain.

$$f'(s) \approx [\phi_1'(s), \ldots, \phi_n'(s)]\, V^{-1}(x, \{\phi_i\}_{i=1}^{n})\, y \qquad y_i = f(x_i)$$

  - Finite-differencing formulas effectively use linear interpolant.

Accuracy of Finite Differences

- Forward and backward differences provide first-order accuracy:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \dots$$

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

Forward difference: $f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$

$$\frac{h}{2} f''(x) + \dots$$

- Centered differencing provides second-order accuracy:

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + O(h^2)$$

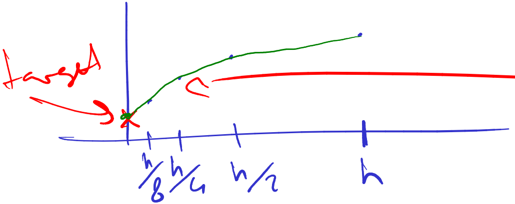
second-order convergence

Extrapolation Techniques

initial guess

$x_0, \ldots, x_k$ ← k iterations of method

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



$$f'(x) = \boxed{\lim_{h \to 0}} \frac{f(x) - f(x-h)}{h}$$

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

given $F(h)$ and $F(h/2)$, would like $F(0)$

$F(h) = a_0 + a_1 h^p + O(h^q)$ where $q > p$, maybe $p+1$ or $p+2$

a_1: $f''(x)/2$ for forward diff.

$$F(h) = a_0 + a_1 h^p + O(h^q)$$

$$F(h/2) = a_0 + a_1 \left(\frac{h}{2}\right)^p + O(h^q)$$

$$a_0 = F(h) - \frac{F(h) - F(h/2)}{1 - \frac{1}{2^p}} + O(h^q)$$

$$= a_0 + a_1 h^p + \underline{O(h^q)} - \frac{\cancel{a_0} + a_1 h^p - \cancel{a_0} - a_1 \left(\frac{h}{2}\right)^p}{1 - \frac{1}{2^p}}$$

$$= a_0 + \cancel{a_1 h^p} - \frac{\cancel{\left(1 - \frac{1}{2^p}\right)} \cancel{a_1 h^p}}{\cancel{1 - \frac{1}{2^p}}} + O(h^q) = a_0 + O(h^q)$$

High-Order Extrapolation

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

Richardson extrapolation

$$
\begin{array}{llllll}
x_0 & & & & & \\
x_1 & x_1^{(1)} & & & & \\
x_2 \to & x_2^{(1)} \to & x_2^{(2)} & & & \\
\vdots & \vdots & \vdots & & & \\
x_{k-1} & x_{k-1}^{(1)} & x_{k-1}^{(2)} & \cdots & x_{(k-1)}^{(k-1)} & \to \text{final guess}
\end{array}
$$

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

For solving nonlinear systems

Aitken Δ^2-process $\to$ Steffensen's method

- quadratic convergence without derivatives
- alternative to Secant method