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 $\mathcal{I}(f) = \int_a^b f(x)dx$:
  - *f is integrable if continuous and bounded.*
  - *Finite number of discontinuities is also often permissible.*
- The condition number of integration is bounded by the distance $b - a$:
  Suppose the input function is perturbed $\hat{f} = f + \delta f$, then

$$\begin{aligned}\delta I &= |\mathcal{I}(\hat{f}) - \mathcal{I}(f)| \\ &\leq |\mathcal{I}(\delta f)| \\ &\leq (b-a)||\delta f||_\infty, \quad \textit{where} \quad \|f\|_\infty = \max_{x\in[a,b]} |f(x)|.\end{aligned}$$

Note that this result does not depend on the magnitude of f or its derivatives, which means integration is generally very well-conditioned, which makes sense since integration corresponds to averaging.

Quadrature Rules

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

$$\mathcal{I}(f) \approx Q_n(f) = \sum_{i=1}^{n} w_i f(x_i)$$

  - $\{x_i\}_{i=1}^n$ *are quadrature nodes or abscissas,* $\{w_i\}_{i=1}^n$ *are quadrature weights.*
  - *Quadrature rule is closed if* $x_1 = a, x_n = b$ *and open otherwise.*
  - *Rule is progressive if nodes of* Q_n *are a subset of those of* Q_{n+1}*.*

- For a fixed set of n nodes, polynomial interpolation followed by integration give $(n-1)$*-degree quadrature rule*:
  - *Accuracy depends on interpolant, is exact for all* $(n-1)$*-degree polynomials.*
  - *Can obtain weights by expressing the unique* $(n-1)$*-degree polynomial interpolant in the Lagrange basis* $p(x) = \sum_{i=1}^n \phi_i(x) f(x_i)$*, so that*

$$Q_n(f) = \mathcal{I}(p) = \sum_{i=1}^{n} \underbrace{\mathcal{I}(\phi_i)}_{w_i} f(x_i),$$

  i.e., weight w_i *is the integral of the* i*th Lagrange basis function.*

Determining Weights for Quadrature Rules

- A quadrature rule provides $\boldsymbol{x}$ and $\boldsymbol{w}$ so as to approximate

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

  Q_n is the integral of the polynomial interpolant p of $(x_1, y_1), \ldots, (x_n, y_n)$.

- *Method of undetermined coefficients* obtains $\boldsymbol{y}$ from *moment equations*, which insure the quadrature rule is exact for all monomials of degree $n-1$:
  - *This also insures the quadrature rule integrates all polynomials of degree up to $n-1$, since by linearity we can decompose the quadrature of any $f = \alpha f_1 + \beta f_2$,*

$$\mathcal{I}(\alpha f_1 + \beta f_2) = \alpha \mathcal{I}(f_1) + \beta \mathcal{I}(f_2)$$

  - *Consider the Vandermonde matrix with nodes at $\boldsymbol{x}$, $\boldsymbol{V}(\boldsymbol{x})$, so $v_{ij} = x_i^{j-1}$. The ith monomial has coefficients given by elementary vector $\boldsymbol{e}^{(i)}$, with $e_i^{(i)} = 1$ and $e_j^{(i)} = 0$ for $j \neq i$, and integral $z_i = \int_0^1 x^{i-1} dx$. From the values of the ith monomial at the nodes, $\boldsymbol{y}^{(i)} = \boldsymbol{V}(\boldsymbol{x})\boldsymbol{e}^{(i)}$, we see that $\boldsymbol{w}$ satisfies*

$$z_i = \langle \boldsymbol{w}, \boldsymbol{y}^{(i)} \rangle = \boldsymbol{w}^T \boldsymbol{V}(\boldsymbol{x}) \boldsymbol{e}^{(i)} \quad \rightarrow \quad \begin{bmatrix} \boldsymbol{e}^{(1)} & \cdots & \boldsymbol{e}^{(n)} \end{bmatrix}^T \boldsymbol{V}(\boldsymbol{x})^T \boldsymbol{w} = \boldsymbol{z}$$

  - *Since $\begin{bmatrix} \boldsymbol{e}^{(1)} & \cdots & \boldsymbol{e}^{(n)} \end{bmatrix} = \boldsymbol{I}$, we obtain $\boldsymbol{w}$ by solving $\boldsymbol{V}(\boldsymbol{x})^T \boldsymbol{w} = \boldsymbol{z}$*

Newton-Cotes Quadrature

Demo: *Newton-Cotes weight finder*

- *Newton-Cotes* quadrature rules are defined by equispaced nodes on $[a, b]$: *open:* $x_i = a + i(b-a)/(n+1)$, *closed:* $x_i = a + (i-1)(b-a)/(n-1)$.
- The *midpoint rule* is the $n = 1$ open Newton-Cotes rule:

$$M(f) = (b-a) f\left(\frac{a+b}{2}\right)$$

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

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

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

$$S(f) = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Error in Newton-Cotes Quadrature

- By our analysis of polynomial quadrature, Newton-cotes rules are exact for polynomials of degree $n-1$, however (1) some, notably the midpoint and Simpson's rule are exact also for degree n, and (2) we also want to understand the error scaling with respect to $b-a$
- 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:

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

Consequently, the midpoint rule is first-degree with third order in $(b-a)$.

Error Estimation

- The trapezoid rule is also just degree 1, since via the prior expansion, $f(m) = f(x) - f'(m)(x-m) - \ldots$, so using $x = a, b$, we get

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

$$\mathcal{I}(f) = T(f) - \underbrace{\frac{f''(m)}{12}(b-a)^3}_{2E(f)} - O((b-a)^5)$$

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

$$T(f) - M(f) \approx 3E(f).$$

Simpson's rule, $S(f) = T(f) + (2/3)(M(f) - T(f))$, thus achieves 5th order accuracy and integrates degree $n = 3$ polynomials exactly.

Error in Polynomial Quadrature Rules

- We can bound the error for a an arbitrary polynomial quadrature rule by applying our error analysis of interpolation,

$$\begin{aligned} |\mathcal{I}(f) - Q_n(f)| &= |\mathcal{I}(f-p)| \\ &\leq (b-a)||f-p||_\infty \\ &\leq \frac{b-a}{4n} h^n ||f^{(n)}||_\infty \\ &= O((b-a)^{n+1}||f^{(n)}||_\infty) \end{aligned}$$

where $h = \max_i(x_{i+1} - x_i)$.

Conditioning of Newton-Cotes Quadrature

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

$$\begin{aligned} |Q_n(\hat{f}) - Q_n(f)| &= |Q_n(\delta f)| \\ &= \sum_{i=1}^{n} w_i \delta f(x_i) \\ &\leq ||\boldsymbol{w}||_1 ||\delta f||_\infty. \end{aligned}$$

Note that we always have $\sum_i w_i = b - a$, since the quadrature rule must be correct for a constant function. So if $\boldsymbol{w}$ is positive $||\boldsymbol{w}||_1 = b - a$, the quadrature rule is stable, i.e. it matches the conditioning of the problem.

- Newton-Cotes quadrature rules have at least one negative weight for any $n \geq 11$: *More generally, $||\boldsymbol{w}||_1 \to \infty$ as $n \to \infty$ for fixed $b - a$. This means that the Newton-Cotes rules can be ill-conditioned.*

Clenshaw-Curtis Quadrature

- To obtain a more stable quadrature rule, we need to ensure the integrated interpolant is well-behaved as n increases:
  - *Chebyshev quadrature nodes ensure that interpolant polynomial has bounded coefficients so long as f is bounded, since the Vandermonde system defining its coefficients is well-conditioned.*
  - *Formally, it can be shown that $w_i > 0$ for the Chebyshev-node (Clenshaw-Curtis) quadrature.*
  - *The weights for Clenshaw-Curtis quadrature rules can be obtained by solutions to Vandermonde systems on $[-1, 1]$ with Chebyshev-spaced nodes, then translating to a desired integration interval.*

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:
  - *Choice of nodes gives additional n parameters for total $2n$ degrees of freedom.*
  - *Permits exact integration of degree-$(2n-1)$ polynomials and corresponding general accuracy.*
- 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 $\boldsymbol{w}$:

  Given any complete basis, we seek to solve the nonlinear equations for $\boldsymbol{x}, \boldsymbol{w}$*,*

$$\boldsymbol{V}(\boldsymbol{x}, \{\phi_i\}_{i=1}^{2n+1})^T \boldsymbol{w} = \boldsymbol{y}, \quad \textit{where} \quad y_i = \mathcal{I}(\phi_i).$$

  - *These nonlinear equations generally have a unique solution* $(\boldsymbol{x}^*, \boldsymbol{w}^*)$*.*
  - *For fixed* $\boldsymbol{x}$*, we have an overdetermined system of linear equations for* $\boldsymbol{w}$*.*

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 $[0, 1]$
  - *We can transform a given integral using variable substitution* $t = \frac{x-a}{b-a}$,

$$\mathcal{I}(f) = \int_a^b f(x)dx = (b-a)\int_0^1 g(t)dt \quad \textit{where} \quad g(t) = f(t(b-a)+a).$$

  - *For quadrature rules defined on* $[-1, 1]$*, we can transform via the substitution* $t = 2\frac{x-a}{b-a} - 1$,

$$\mathcal{I}(f) = \int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^1 g(t)dt \quad \textit{where} \quad g(t) = f((t+1)(b-a)/2+a).$$

- Gaussian quadrature rules are accurate and stable but not progressive (nodes cannot be reused to obtain higher-degree approximation):
  - *maximal degree is obtained*
  - *weights are always positive (perfect conditioning)*

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:
  - *Gaussian quadrature rule G_{2n+1} would use same number of points and have degree $4n+1$.*
  - *Kronod rule points are optimal chosen to reuse all points of G_n, so $n+1$ rather than $2n+1$ new evaluations are necessary.*
- *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-Radau uses one of two end-points as a node, while Gauss-Lobatto quadrature uses both.

Composite and Adaptive Quadrature

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

  For example, we can derive simple composite Newton-Cotes rules by partitioning the domain into sub-intervals $[x_i, x_{i+1}]$:

  - *composite midpoint rule*

$$\mathcal{I}(f) = \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx \approx \sum_{i=1}^{n-1} (x_{i+1} - x_i) f((x_{i+1} + x_i)/2)$$

  - *composite trapezoid rule*

$$\mathcal{I}(f) = \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx \approx \sum_{i=1}^{n-1} \frac{(x_{i+1} - x_i)}{2} (f(x_{i+1}) + f(x_i))$$

- Composite quadrature can be done with adaptive refinement:

  Introduce new nodes where error estimate is large. Error estimate can be obtained by e.g., comparing trapezoid and midpoint rules, but can be completely wrong if function is insufficiently smooth.

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.

  Composite multidimensional rules are also possible by partitioning the domain into chunks, e.g., triangulation.
- High-dimensional integration is often effectively done by *Monte Carlo*:

$$\int_\Omega f(\boldsymbol{x})d\boldsymbol{x} = E[Y], \quad Y = \frac{|\Omega|}{N}\sum_{i=1}^{N} Y_i, \quad Y_i = f(\boldsymbol{x}_i), \quad \boldsymbol{x}_i \text{ chosen randomly from } \Omega.$$

  - *Convergence rate is independent of function (effective polynomial degree approximation) or dimension of integration domain.*
  - *Instead, it depends on number of samples (N), with error scaling as $O(1/\sqrt{N})$.*

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

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

  - *Useful for recovering signal u given response function with kernel K and measurements of f.*
  - *Many differential equation problems can be transformed integral equations.*
- Using a quadrature rule with weights $w_1, \ldots, w_n$ and nodes $t_1, \ldots, t_n$ obtain

$$\sum_{j=1}^{n} w_j K(s,t_j)u(t_j) = f(s).$$

Discrete sample of f on $s_1, \ldots, s_n$ yields a linear system of equations,

$$\sum_{j=1}^{n} w_j K(s_i,t_j)u(t_j) = f(s_i).$$

Numerical Differentiation

- Automatic (symbolic) differentiation is a surprisingly viable option:
  - *Any straight-line (no data-dependent branches) computer program is directly differentiable, when a derivative exists, since it is an assembly of basic arithmetic operations.*
  - *Existing software packages can automatically differentiate whole programs.*
- 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,*

$$f'(x) \approx p'(x) = \begin{bmatrix} \phi_1'(x) & \cdots & \phi_n'(x) \end{bmatrix}^T \boldsymbol{V}(\boldsymbol{t}, \{\phi_i\}_{i=1}^n)^{-1} \boldsymbol{y}, \text{ where } y_i = f(t_i).$$

  - *Obtaining the values of the derivative at the interpolation nodes, can be done via*

$$\underbrace{\boldsymbol{V}(\boldsymbol{t}, \{\phi_i'\}_{i=1}^n) \boldsymbol{V}(\boldsymbol{t}, \{\phi_i\}_{i=1}^n)^{-1}}_{\text{Differentiation matrix}} \boldsymbol{y}, \text{ where } y_i = f(t_i).$$

  - *Finite-differencing formulas effectively use piecewise linear interpolant.*

Accuracy of Finite Differences

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

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

  These can be derived, respectively from forward and backward Taylor expansions of f about x,

$$f(x+h) = f(x) + f'(x)h + f''(x)h^2/2 + \dots$$
$$f(x-h) = f(x) - f'(x)h + f''(x)h^2/2 - \dots$$

  For forward differencing, we obtain an approximation from the first equation,

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

- *Centered differencing* provides second-order accuracy. *Subtracting the backward Taylor expansion from the forward, we obtain centered differencing,*

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

  Second order accuracy is due to cancellation of odd terms like $f''(x)h/2$.

Extrapolation Techniques

Demo: *Richardson with Finite Differences*

- Given a sequence of approximations to the result of a smooth function, a more accurate approximation may be obtained by *extrapolating* this series.

  As we lower the step size h in a finite-difference formula, we can try to extrapolate the series $F(h), F(h/2), F(h/4), F(h/8), \ldots$ to $h = 0$. We have

  $$F(h) = a_0 + a_1 h^p + O(h^r) \text{ as } h \to 0 \text{ and seek to determine } F(0) = a_0,$$

  for example in centered differences $p = 2$ and $r = 4$.

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

  Seek to eliminate $a_1 h^p$ term in $F(h)$, $F(h/2)$ to improve approximation of a_0,

$$\begin{aligned} F(h) &= a_0 + a_1 h^p + O(h^r), \\ F(h/2) &= a_0 + a_1 h^p / 2^p + O(h^r), \\ a_0 &= F(h) - \frac{F(h) - F(h/2)}{1 - 1/2^p} + O(h^r). \end{aligned}$$

High-Order Extrapolation

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

  Compute k composite quadrature rules with node spacing $(b-a)/2^i$ for $i \in, 1 \ldots, k$, resulting in $I_1, \ldots, I_k$ then apply Richardson extrapolation first to to (I_j, I_{j+1}) for each $j \in \{1, \ldots, k-1\}$ to obtain $k-1$ approximations and so-on.

- Extrapolation can be used within an iterative procedure at each step:
  For example, Steffensen's method for finding roots of nonlinear equations,

$$x_{n+1} = x_n + \frac{f(x_n)}{1 - f(x_n + f(x_n))/f(x_n)},$$

  derived from Aitken's delta-squared extrapolation process:

  - *achieves quadratic convergence,*
  - *requires no derivative,*
  - *competes with the Secant method (quadratic versus superlinear convergence, but an extra function evaluation necessary).*