Lecture Notes to Accompany

Scientific Computing

An Introductory Survey

Second Edition

by Michael T. Heath

Chapter 8

Numerical Integration and Differentiation

Integration

For $f\colon \mathbb{R} \to \mathbb{R}$, definite integral over interval $[a, b]$,

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

defined by limit of Riemann sums

$$R_n = \sum_{i=1}^{n} (x_{i+1} - x_i)\, f(\xi_i)$$

Riemann integral exists provided integrand f is bounded and continuous almost everywhere

Absolute condition number of integration with respect to perturbations in integrand is $b - a$

Integration inherently well-conditioned because of smoothing effect

Numerical Quadrature

Quadrature rule is weighted sum of finite number of sample values of integrand function

Main issues:

- How to choose sample points
- How to weight their contributions

to obtain desired level of accuracy at low cost

Computational work measured by number of evaluations of integrand function required

Quadrature Rules

An n-point quadrature rule has form

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

- points x_i called *nodes* or *abscissas*
- multipliers w_i called *weights*

Quadrature rule *open* if $a < x_1$ and $x_n < b$, *closed* if $a = x_1$ and $x_n = b$

Quadrature Rules, continued

Quadrature rules based on polynomial interpolation:

- Integrand function f sampled at finite set of points
- Polynomial interpolant at those points is determined
- Integral of interpolant taken as estimate for integral of original function

In practice, interpolating polynomial not determined explicitly but used to determine weights corresponding to nodes

If Lagrange interpolation used, then weights given by

$$w_i = \int_a^b \ell_i(x), \quad i = 1, \ldots, n$$

Method of Undetermined Coefficients

Alternative derivation of quadrature rule uses *method of undetermined coefficients*

To derive n-point rule on interval $[a, b]$, take nodes $x_1, \ldots, x_n$ as given and consider weights $w_1, \ldots, w_n$ as coefficients to be determined

Force quadrature rule to integrate first n polynomial basis functions exactly, and by linearity, it will integrate any polynomial of degree $n-1$ exactly

Thus obtain system of *moment* equations that determines weights for quadrature rule

Example: Undetermined Coefficients

Derive three-point rule

$$Q_3(f) = w_1 f(x_1) + w_2 f(x_2) + w_3 f(x_3)$$

on interval $[a, b]$ using monomial basis

As nodes, take $x_1 = a$, $x_2 = (a+b)/2$, and $x_3 = b$; first three monomials are 1, x, and x^2

Resulting system of moment equations is

$$w_1 \cdot 1 + w_2 \cdot 1 + w_3 \cdot 1 = \int_a^b 1\,dx = x|_a^b = b - a,$$

$$w_1 \cdot a + w_2 \cdot (a+b)/2 + w_3 \cdot b =$$

$$\int_a^b x\,dx = (x^2/2)|_a^b = (b^2 - a^2)/2,$$

$$w_1 \cdot a^2 + w_2 \cdot ((a+b)/2)^2 + w_3 \cdot b^2 =$$

$$\int_a^b x^2\,dx = (x^3/3)|_a^b = (b^3 - a^3)/3$$

Example Continued

In matrix form,

$$\begin{bmatrix} 1 & 1 & 1 \\ a & (a+b)/2 & b \\ a^2 & ((a+b)/2)^2 & b^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} b-a \\ (b^2-a^2)/2 \\ (b^3-a^3)/3 \end{bmatrix}$$

Solving system by Gaussian elimination, obtain weights

$$w_1 = \frac{b-a}{6}, \quad w_2 = \frac{2(b-a)}{3}, \quad w_3 = \frac{b-a}{6},$$

which is known as Simpson's rule

Method of Undetermined Coefficients

More generally, *Vandermonde* system

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} b-a \\ (b^2-a^2)/2 \\ \vdots \\ (b^n-a^n)/n \end{bmatrix}$$

determines weights w_i for any n and choice of nodes x_i

Accuracy of Quadrature Rules

Quadrature rule is of *degree* d if it is exact for every polynomial of degree d, but not exact for some polynomial of degree $d + 1$

By construction, n-point interpolatory quadrature rule is of degree at least $n - 1$

Rough error bound

$$|I(f) - Q_n(f)| \le \frac{1}{4} h^{n+1} \|f^{(n)}\|_\infty,$$

where $h = \max\{x_{i+1} - x_i : i = 1, \ldots, n - 1\}$, shows that

$$Q_n(f) \to I(f)$$

as $n \to \infty$, provided $f^{(n)}$ remains well behaved

Can obtain higher accuracy by increasing n or decreasing h

Progressive Quadrature Rules

Sequence of quadrature rules is *progressive* if nodes of Q_{n_1} are subset of nodes of Q_{n_2} for $n_2 > n_1$

For progressive rules, function evaluations used in one rule can be reused in another, reducing overall cost

To attain higher accuracy, can increase number of points n, or subdivide interval into smaller subintervals

In either case, efficiency enhanced if successive rules are progressive

Stability of Quadrature Rules

Absolute condition number of quadrature rule is sum of magnitudes of weights,

$$\sum_{i=1}^{n} |w_i|$$

If weights are all nonnegative, then absolute condition number of quadrature rule is $b - a$, same as that of underlying integral, so rule is stable

If any weights are negative, then absolute condition number can be much larger, and rule can be unstable

Newton-Cotes Quadrature

Newton-Cotes quadrature rules use equally spaced nodes in interval $[a, b]$

Examples:

- *Midpoint* rule:

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

- *Trapezoid* rule:

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

- *Simpson's* rule:

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

Example: Newton-Cotes Quadrature

Approximate integral

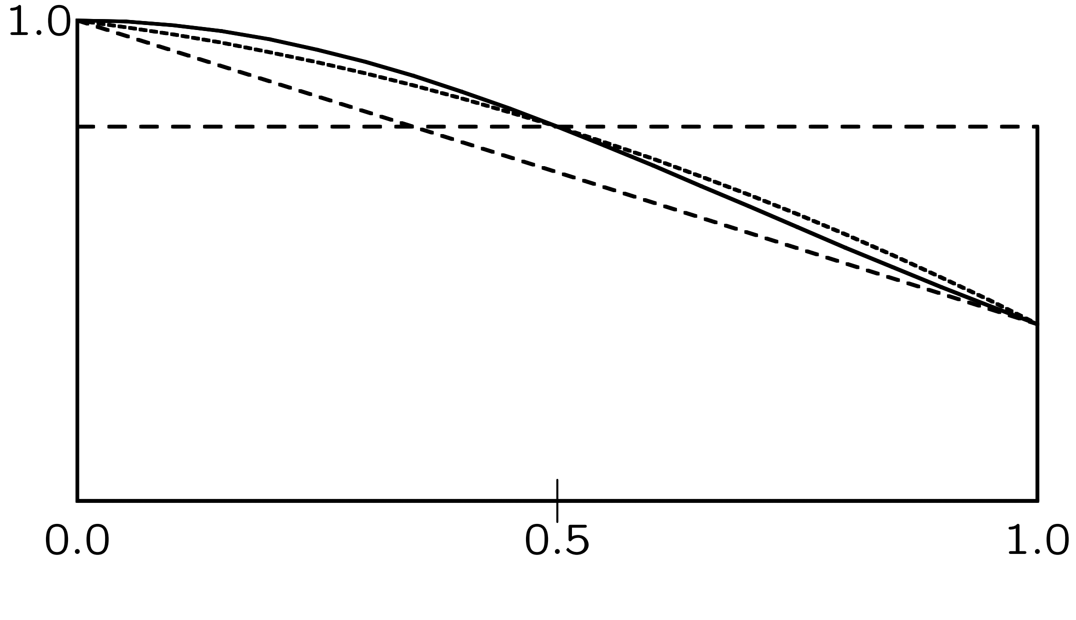
$$I(f) = \int_0^1 \exp(-x^2)\,dx \approx 0.746824$$

$$M(f) = (1-0)\exp(-1/4) \approx 0.778801$$

$$T(f) = (1/2)[\exp(0) + \exp(-1)] \approx 0.683940$$

$$S(f) = (1/6)[\exp(0) + 4\exp(-1/4) + \exp(-1)]$$

$$\approx 0.747180$$



Error Estimation

Expanding integrand f in Taylor series about midpoint $m = (a+b)/2$ of interval $[a, b]$,

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

$$+\frac{f'''(m)}{6}(x-m)^3 + \frac{f^{(4)}(m)}{24}(x-m)^4 + \cdots$$

Integrating from a to b, odd-order terms drop out, yielding

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

$$+\frac{f^{(4)}(m)}{1920}(b-a)^5 + \cdots$$

$$= M(f) + E(f) + F(f) + \cdots,$$

where $E(f)$ and $F(f)$ represent first two terms in error expansion for midpoint rule

Error Estimation, continued

If we substitute $x = a$ and $x = b$ into Taylor series, add two series together, observe once again that odd-order terms drop out, solve for $f(m)$, and substitute into midpoint formula, we obtain

$$I(f) = T(f) - 2E(f) - 4F(f) - \cdots$$

Thus, provided length of interval is sufficiently small and $f^{(4)}$ is well behaved, midpoint rule is about twice as accurate as trapezoid rule

Halving length of interval decreases error in either rule by factor of about 1/8

Error Estimation, continued

Difference between midpoint and trapezoid rules provides estimate for error in either of them:

$$T(f) - M(f) = 3E(f) + 5F(f) + \cdots,$$

so

$$E(f) \approx \frac{T(f) - M(f)}{3}$$

Weighted combination of midpoint and trapezoid rules eliminates $E(f)$ term from error expansion:

$$\begin{aligned} I(f) &= \frac{2}{3}M(f) + \frac{1}{3}T(f) - \frac{2}{3}F(f) + \cdots \\ &= S(f) - \frac{2}{3}F(f) + \cdots, \end{aligned}$$

which gives alternate derivation for Simpson's rule and estimate for its error

Example: Error Estimation

We illustrate error estimation by computing approximate value for integral $\int_0^1 x^2\,dx$

$$M(f) = (1-0)(1/2)^2 = 1/4,$$

$$T(f) = \frac{1-0}{2}(0^2 + 1^2) = 1/2,$$

$$E(f) \approx (T(f) - M(f))/3 = (1/4)/3 = 1/12$$

So error in $M(f)$ is about 1/12, and error in $T(f)$ is about $-1/6$

Also,

$$S(f) = (2/3)M(f) + (1/3)T(f)$$

$$= (2/3)(1/4) + (1/3)(1/2) = 1/3,$$

which is exact for this integral, as expected

Accuracy of Newton-Cotes Quadrature

Since n-point Newton-Cotes rule is based on polynomial interpolant of degree $n-1$, we expect rule to have degree $n-1$

Thus, we expect midpoint rule to have degree zero, trapezoid rule degree one, Simpson's rule degree two, and so on

From Taylor series expansion, error for midpoint rule depends on second and higher derivatives of integrand, which vanish for linear as well as constant polynomials

So midpoint rule integrates linear polynomials exactly, and its degree is one rather than zero

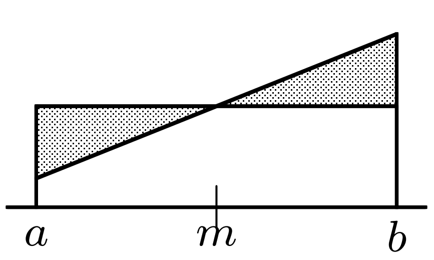
Similarly, error for Simpson's rule depends on fourth and higher derivatives, which vanish for cubics as well as quadratic polynomials, so Simpson's rule is of degree three

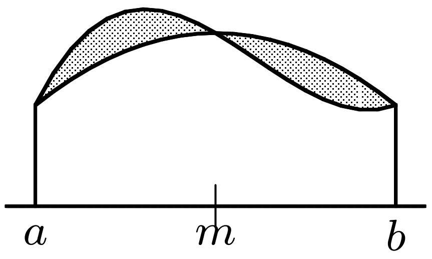
Accuracy of Newton-Cotes Quadrature

In general, odd-order Newton-Cotes rule gains extra degree beyond that of polynomial interpolant on which it is based

n-point Newton-Cotes rule is of degree $n-1$ if n is even, but of degree n if n is odd

This phenomenon is due to *cancellation* of positive and negative errors:





Drawbacks of Newton-Cotes Rules

Newton-Cotes quadrature rules are simple and often effective, but they have drawbacks:

- Using large number of equally spaced nodes may incur erratic behavior associated with high-degree polynomial interpolation (e.g., weights may be negative)
- Indeed, every n-point Newton-Cotes rule with $n \geq 11$ has at least one negative weight, and $\sum_{i=1}^{n} |w_i| \to \infty$ as $n \to \infty$, so Newton-Cotes rules become arbitrarily ill-conditioned
- Newton-Cotes rules are not of highest degree possible for number of nodes used

Clenshaw-Curtis Quadrature

As with polynomial interpolation, use of *Chebyshev points* produces better results

Improved accuracy results from good approximation properties of interpolation at Chebyshev points

Weights are always positive and approximate integral always converges to exact integral as $n \to \infty$

Quadrature rules based on Chebyshev points can be implemented very efficiently, as shown by Clenshaw and Curtis

So Clenshaw-Curtis quadrature has many attractive features, but still does not have maximum possible degree for number of nodes used

Gaussian Quadrature

Gaussian rules are based on polynomial interpolation, but nodes as well as weights are chosen to maximize degree of resulting rule

With $2n$ parameters, we can obtain degree of $2n - 1$

Gaussian quadrature rules can be derived by method of undetermined coefficients, but resulting system of moment equations that determines nodes and weights is nonlinear

Also, nodes are usually irrational, even if endpoints of interval are rational

Although inconvenient for hand computation, nodes and weights are tabulated in advance and stored in subroutine for use on computer

Example: Gaussian Quadrature Rule

Derive two-point Gaussian rule on $[-1,1]$:

$$G_2(f) = w_1 f(x_1) + w_2 f(x_2),$$

where nodes x_i and weights w_i are chosen to maximize degree of resulting rule

We use method of undetermined coefficients, but now nodes as well as weights are unknown parameters to be determined

Four parameters are to be determined, so we expect to be able to integrate cubic polynomials exactly, since cubics depend on four parameters

Example Continued

Requiring rule to integrate first four monomials exactly gives moment equations:

$$w_1 + w_2 = \int_{-1}^{1} 1\,dx = x|_{-1}^{1} = 2,$$

$$w_1 x_1 + w_2 x_2 = \int_{-1}^{1} x\,dx = (x^2/2)|_{-1}^{1} = 0,$$

$$w_1 x_1^2 + w_2 x_2^2 = \int_{-1}^{1} x^2\,dx = (x^3/3)|_{-1}^{1} = 2/3,$$

$$w_1 x_1^3 + w_2 x_2^3 = \int_{-1}^{1} x^3\,dx = (x^4/4)|_{-1}^{1} = 0$$

One solution of this system of four nonlinear equations in four unknowns is given by

$$x_1 = -1/\sqrt{3}, \quad x_2 = 1/\sqrt{3}, \quad w_1 = 1, \quad w_2 = 1,$$

and other solution is obtained by reversing signs of x_1 and x_2

Example Continued

Resulting two-point Gaussian rule has form

$$G_2(f) = f(-1/\sqrt{3}) + f(1/\sqrt{3}),$$

and by construction it has degree three

In general, for each n there is unique n-point Gaussian rule, and it is of degree $2n-1$

Gaussian quadrature rules can also be derived using orthogonal polynomials

Change of Interval

Gaussian rules are somewhat more difficult to apply than Newton-Cotes rules because weights and nodes are usually derived for some specific interval, such as $[-1, 1]$

Given interval of integration $[a, b]$ must be transformed into standard interval for which nodes and weights have been tabulated

To use quadrature rule tabulated on interval $[\alpha, \beta]$,

$$\int_\alpha^\beta f(x)\,dx \approx \sum_{i=1}^n w_i f(x_i),$$

to approximate integral on interval $[a, b]$,

$$I(g) = \int_a^b g(t)\,dt,$$

we must change variable from x in $[\alpha, \beta]$ to t in $[a, b]$

Change of Interval, continued

Many transformations are possible, but simple linear transformation

$$t = [(b-a)x + a\beta - b\alpha]/(\beta - \alpha)$$

has advantage of preserving degree of quadrature rule

Gaussian Quadrature

Gaussian quadrature rules have maximal degree and optimal accuracy for number of nodes used

Weights are always positive and approximate integral always converges to exact integral as $n \to \infty$

Unfortunately, Gaussian rules of different orders have no nodes in common (except possibly midpoint), so Gaussian rules are *not* progressive

Thus, estimating error using Gaussian rules of different order requires evaluating integrand function at full set of nodes of both rules

Progressive Gaussian Quadrature

Avoiding this additional work is motivation for *Kronrod* quadrature rules

Such rules come in pairs, n-point Gaussian rule G_n, and $(2n+1)$-point Kronrod rule K_{2n+1}, whose nodes are optimally chosen subject to constraint that all nodes of G_n are reused in K_{2n+1}

$(2n+1)$-point Kronrod rule is of degree $3n+1$, whereas true $(2n+1)$-point Gaussian rule would be of degree $4n+1$

In using Gauss-Kronrod pair, value of K_{2n+1} is taken as approximation to integral, and error estimate is given by

$$(200|G_n - K_{2n+1}|)^{1.5}$$

Progressive Gaussian Quadrature, cont.

Because they efficiently provide high accuracy and reliable error estimate, Gauss-Kronrod rules are among most effective methods for numerical quadrature

They form basis for many quadrature routines available in major software libraries

Pair (G_7, K_{15}) is commonly used standard

Patterson quadrature rules further extend this idea by adding $2n+2$ optimally chosen nodes to $2n+1$ nodes of Kronrod rule K_{2n+1}, yielding progressive rule of degree $6n+5$

Gauss-Radau and Gauss-Lobatto rules specify one or both endpoints, respectively, as nodes and then choose remaining nodes and all weights to maximize degree

Composite Quadrature

Alternative to using more nodes and higher degree rule is to subdivide original interval into subintervals, then apply simple quadrature rule in each subinterval

Summing partial results then yields approximation to overall integral

Equivalent to using piecewise interpolation to derive *composite* quadrature rule over interval

Composite rule is always stable if underlying simple rule is stable

Approximate integral converges to exact interval as number of subintervals goes to infinity provided underlying simple rule has degree at least zero

Examples: Composite Quadrature

If interval $[a, b]$ is subdivided into k subintervals of length $h = (b - a)/k$ and $x_j = a + jh$, $j = 0, \ldots, k$, then *composite midpoint rule* is given by

$$\begin{aligned} M_k(f) &= \sum_{j=1}^{k} (x_j - x_{j-1})\, f\left(\frac{x_{j-1} + x_j}{2}\right) \\ &= h \sum_{j=1}^{k} f\left(\frac{x_{j-1} + x_j}{2}\right), \end{aligned}$$

and *composite trapezoid rule* is given by

$$T_k(f) = \sum_{j=1}^{k} \frac{(x_j - x_{j-1})}{2} \big(f(x_{j-1}) + f(x_j)\big)$$

$$= h\, \left(\tfrac{1}{2} f(a) + f(x_1) + \cdots + f(x_{k-1}) + \tfrac{1}{2} f(b)\right)$$

Composite Quadrature Rules, cont.

Composite quadrature offers simple means of estimating error by using two different levels of subdivision, which is easily progressive

For example, halving interval length reduces error in midpoint or trapezoid rule by factor of about 1/8

Halving width of each subinterval means twice as many subintervals are required, so overall reduction in error is by factor of about 1/4

If h denotes subinterval length, then dominant term in error of composite midpoint or trapezoid rules is $\mathcal{O}(h^2)$

Dominant term in error of composite Simpson's rule is $\mathcal{O}(h^4)$, so halving subinterval length reduces error by factor of about 1/16

Adaptive Quadrature

Composite quadrature rule with error estimate suggests simple *automatic* quadrature procedure:

Continue to subdivide all subintervals, say by half, until overall error estimate falls below desired tolerance

Such uniform subdivision is grossly inefficient for many integrands

More intelligent approach is *adaptive* quadrature, in which domain of integration is selectively refined to reflect behavior of particular integrand function

Adaptive Quadrature, continued

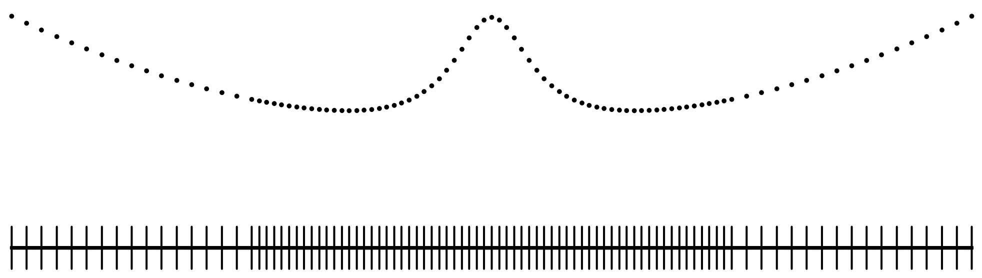
Start with pair of quadrature rules whose difference gives error estimate

Apply both rules on initial interval $[a, b]$

If difference between rules exceeds error tolerance, subdivide interval and apply rules in each subinterval

Continue subdividing subintervals, as necessary, until tolerance is met on all subintervals

Integrand is sampled densely in regions where it is difficult to integrate and sparsely in regions where it is easy



Adaptive Quadrature, continued

Adaptive quadrature tends to be effective in practice, but it can be fooled: both approximate integral and error estimate can be completely wrong

Integrand function is sampled at only finite number of points, so significant features of integrand may be missed

For example, interval of integration may be very wide but "interesting" behavior of integrand is confined to narrow range

Sampling by automatic routine may miss interesting part of integrand behavior, and resulting value for integral may be completely wrong

Adaptive Quadrature, continued

Adaptive quadrature routine may be inefficient in handling discontinuities in integrand

For example, adaptive routine may use many function evaluations refining region around discontinuity of integrand

To prevent this, call quadrature routine separately to compute integral on either side of discontinuity, avoiding need to resolve discontinuity

Integrating Tabular Data

If integrand is defined only by table of its values at discrete points, then reasonable approach is to integrate piecewise interpolant

For example, integrating piecewise linear interpolant to tabular data gives composite trapezoid rule

Excellent method for integrating tabular data is to use Hermite cubic or cubic spline interpolation

In effect, overall integral is computed by integrating each of cubic pieces that make up interpolant

This facility is provided by many spline interpolation packages

Improper Integrals

To compute integrals over infinite or semi-infinite intervals, several approaches are possible:

- Replace infinite limits of integration by carefully chosen finite values
- Transform variable of integration so that new interval is finite, but care must be taken not to introduce singularities
- Use quadrature rule designed for infinite interval

Double Integrals

Some approaches for evaluating double integrals include:

- Use automatic one-dimensional quadrature routine for each dimension, one for outer integral and another for inner integral
- Use product quadrature rule resulting from applying one-dimensional rule to successive dimensions
- Use non-product quadrature rule for regions such as triangles

Multiple Integrals

To evaluate multiple integrals in higher dimensions, only generally viable approach is Monte Carlo method

Function is sampled at n points distributed randomly in domain of integration, and mean of function values is multiplied by area (or volume, etc.) of domain to obtain estimate for integral

Error in estimate goes to zero as $1/\sqrt{n}$, so to gain one additional decimal digit of accuracy requires n to increase by factor of 100

For this reason, Monte Carlo calculations of integrals often require millions of evaluations of integrand

Multiple Integrals, continued

Monte Carlo method is not competitive for dimensions one or two, but beauty of method is that its convergence rate is independent of number of dimensions

For example, one million points in six dimensions amounts to only ten points per dimension, which is much better than any type of conventional quadrature rule would require for same level of accuracy

Integral Equations

Typical integral equation has form

$$\int_a^b K(s,t)u(t)\,dt = f(s),$$

where *kernel* K and right-hand side f are known functions, and unknown function u is to be determined

Solved numerically by discretizing variables and replacing integral by quadrature formula:

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

System of linear algebraic equations $\boldsymbol{Ax} = \boldsymbol{y}$, where $a_{ij} = w_j K(s_i,t_j)$, $y_i = f(s_i)$, and $x_j = u(t_j)$, is solved for $\boldsymbol{x}$ to obtain discrete sample of approximate values of function u

Integral Equations, continued

Though straightforward to solve formally, integral equations are often extremely sensitive to perturbations in input data, which are often subject to random experimental or measurement errors

Resulting linear system is highly ill-conditioned

Techniques for coping with ill-conditioning include:

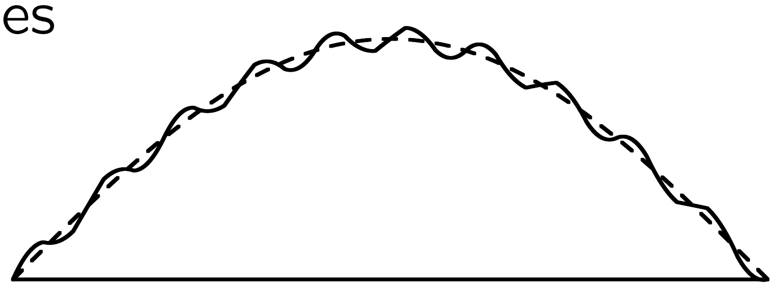
- Truncated SVD
- Regularization
- Constrained optimization

Numerical Differentiation

Differentiation is inherently sensitive, as small perturbations in data can cause large changes in result

Differentiation is inverse of integration, which is inherently stable because of smoothing effect

For example, two functions shown below have very similar definite integrals but very different derivatives



Numerical Differentiation, continued

To approximate derivative of function whose values are known only at discrete set of points, good approach is to fit some smooth function to given data and then differentiate approximating function

If given data are sufficiently smooth, then interpolation may be appropriate, but if data are noisy, then smoothing approximating function, such as least squares spline, is more appropriate

Finite Difference Approximations

Given smooth function $f\colon \mathbb{R} \to \mathbb{R}$, we wish to approximate its first and second derivatives at point x

Consider Taylor series expansions

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \cdots,$$

and

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{6}h^3 + \cdots$$

Solving for $f'(x)$ in first series, obtain *forward difference formula*

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x)}{h} - \frac{f''(x)}{2}h + \cdots \\ &\approx \frac{f(x+h) - f(x)}{h}, \end{aligned}$$

which is first-order accurate since dominant term in remainder of series is $\mathcal{O}(h)$

Finite Difference Approximations, cont.

Similarly, from second series derive *backward difference formula*

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \frac{f''(x)}{2}h + \cdots$$
$$\approx \frac{f(x) - f(x-h)}{h},$$

which is also first-order accurate

Subtracting second series from first series gives *centered difference formula*

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f'''(x)}{6}h^2 + \cdots$$
$$\approx \frac{f(x+h) - f(x-h)}{2h},$$

which is second-order accurate

Finite Difference Approximations, cont.

Adding both series together gives centered difference formula for second derivative

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

$$-\frac{f^{(4)}(x)}{12}h^2 + \cdots,$$

$$\approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2},$$

which is also second-order accurate

Finite difference formulas can also be derived by polynomial interpolation, which is less cumbersome than Taylor series for higher-order accuracy or higher-order derivatives, and is more easily generalized to unequally spaced points

Automatic Differentiation

Computer program expressing function is composed of basic arithmetic operations and elementary functions, each of whose derivatives is easily computed

Derivatives can be propagated through program by repeated use of chain rule, computing derivative of function step by step along with function itself

Result is true derivative of original function, subject only to rounding error but suffering no discretization error

Software packages are available implementing this *automatic differentiation* (AD) approach

Richardson Extrapolation

In many problems, such as numerical integration or differentiation, approximate value for some quantity is computed based on some step size

Ideally, we would like to obtain limiting value as step size approaches zero, but we cannot take step size arbitrarily small because of excessive cost or rounding error

Based on values for nonzero step sizes, however, we may be able to estimate what value would be for step size of zero

Richardson Extrapolation, continued

Let $F(h)$ denote value obtained with step size h

If we compute value of F for some nonzero step sizes, and if we know theoretical behavior of $F(h)$ as $h \to 0$, then we can extrapolate from known values to obtain approximate value for $F(0)$

Suppose that

$$F(h) = a_0 + a_1 h^p + \mathcal{O}(h^r)$$

as $h \to 0$ for some p and r, with $r > p$

Assume we know values of p and r, but not a_0 or a_1 (indeed, $F(0) = a_0$ is what we seek)

Richardson Extrapolation, continued

Suppose we have computed F for two step sizes, say h and h/q for some positive integer q

Then we have

$$F(h) = a_0 + a_1 h^p + \mathcal{O}(h^r)$$

and

$$\begin{aligned} F(h/q) &= a_0 + a_1 (h/q)^p + \mathcal{O}(h^r) \\ &= a_0 + a_1 q^{-p} h^p + \mathcal{O}(h^r) \end{aligned}$$

This system of two linear equations in two unknowns a_0 and a_1 is easily solved to obtain

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

Accuracy of improved value, a_0, is $\mathcal{O}(h^r)$

Richardson Extrapolation, continued

Extrapolated value, though improved, is still only approximation, not exact solution, and its accuracy is still limited by step size and arithmetic precision used

If $F(h)$ is known for several values of h, then extrapolation process can be repeated to produce still more accurate approximations, up to limitations imposed by finite-precision arithmetic

Example: Richardson Extrapolation

We use Richardson extrapolation to improve accuracy of finite difference approximation to derivative of function sin(x) at $x = 1$

Using first-order accurate forward difference formula, we have

$$F(h) = a_0 + a_1 h + \mathcal{O}(h^2),$$

so $p = 1$ and $r = 2$ in this instance

Using step sizes of $h = 0.5$ and $h/2 = 0.25$ (i.e., $q = 2$), we obtain

$$F(h) = \frac{\sin(1.5) - \sin(1)}{0.5} = 0.312048$$

and

$$F(h/2) = \frac{\sin(1.25) - \sin(1)}{0.25} = 0.430055$$

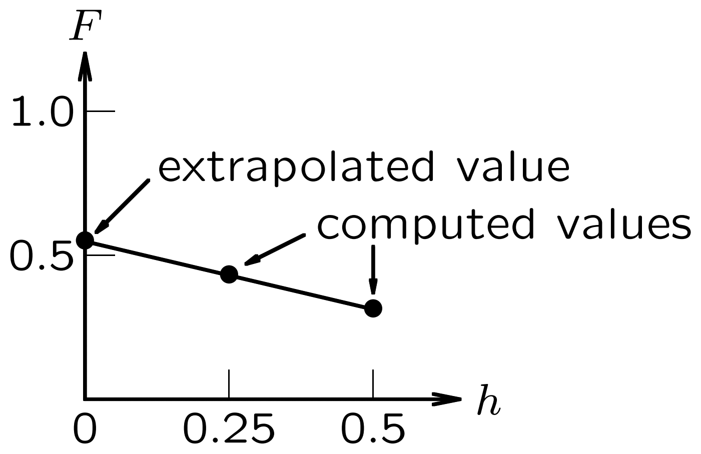
Example Continued

Extrapolated value is then given by

$$F(0) = a_0 = F(h) + \frac{F(h) - F(h/2)}{(1/2) - 1}$$

$$= 2F(h/2) - F(h) = 0.548061$$

For comparison, correctly rounded result is given by $\cos(1) = 0.540302$



Example: Romberg Integration

As another example, we evaluate

$$\int_0^{\pi/2} \sin(x)\, dx$$

If we use composite trapezoid rule, then we have

$$F(h) = a_0 + a_1 h^2 + \mathcal{O}(h^4),$$

so $p = 2$ and $r = 4$ in this instance

With $h = \pi/2$, $F(h) = F(\pi/2) = 0.785398$

With $q = 2$, $F(h/2) = F(\pi/4) = 0.948059$

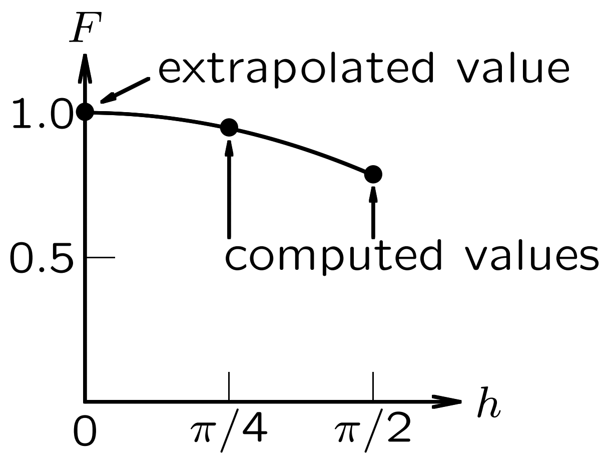
Example Continued

Extrapolated value is then given by

$$F(0) = a_0 = F(h) + \frac{F(h) - F(h/2)}{2^{-2} - 1}$$

$$= \frac{4F(h/2) - F(h)}{3} = 1.002280,$$

which is substantially more accurate than values previously computed (exact answer is 1)



Romberg Integration

Continued Richardson extrapolations using composite trapezoid rule with successively halved step sizes is called *Romberg* integration

It is capable of producing very high accuracy (up to limit imposed by arithmetic precision) for very smooth integrands

It is often implemented in automatic (though nonadaptive) fashion, with extrapolations continuing until change in successive values falls below specified error tolerance