Numerical Integration Summary

Larry Caretto
Mechanical Engineering 309

Numerical Analysis of Engineering Systems

April 16, 2014

California State University Northridge

Outline

- Basic Integration Formulas
  - Simpson's Rule and Trapezoid Rule
- Halving step size to get desired error
- Review Richardson Extrapolation
- Review Romberg Integration
- MATLAB function for quadrature
- Application of Romberg Integration
- Programming Assignment Six

California State University Northridge 2

Numerical integration formulas

- Trapezoid Rule

$$I=\int_a^b f(x)dx = T+E = h\left[\frac{f_0+f_N}{2}+\sum_{i=1}^{N-1} f_i\right]+O(h^2)$$

- Simpson's (1/3) rule (even N only)

$$I=\int_a^b f(x)dx = S+E = \frac{h}{3}\left[f_0+f_N+4\sum_{i=1,3,5}^{N-1} f_i+2\sum_{i=2,4,6}^{N-2} f_i\right]+O(h^4)$$

- Basic definitions of step size, h, number of intervals, N, $x_0 = a$, $x_N = b$, $f_k = f(a + kh)$

$$h=\frac{b-a}{N} \qquad x_k = a+kh \qquad f_k = f(x_k)$$

California State University Northridge 3

Doubling N for Accuracy

- The formulas for the Trapezoid Rule and Simpson's rule evaluate the integrand, f(x) at evenly spaced intervals $x_k = a + kh$
- When N is doubled, we can use integrand evaluations from previous value of N
- For trapezoid rule, $T(h) = T(2h)/2 + h\sum_{odd\ k} f(a+kh)$
- For Simpson's rule must consider terms multiplied by 1, 2, and 4 separately

California State University Northridge 4

Repeated Trapezoid Rule

```
cvg = false : N = 1 : maxN = 1E6
trap = (f(a) + f(b)) * (b - a) / 2
While not cvg And N < maxN
    oldT = trap : sum = 0
    N = 2 * N : h = (b - a) / N
    for k = 1 to N - 1 Step 2
        sum = sum + f(a + k * h)
    end for
    trap = h * sum + oldT/2
    cvg = abs(trap - oldT) <=
            maxRelErr * abs(trap)
Loop
```

California State University Northridge

Test cvg after end of while loop

5

Repeated Simpson's Rule 1/2

```
cvg = false : N = 1 : maxN = 1E6
maxRelErr = 1E-10
ends = f(a) + f(b) : evens = 0
odds = f((a + b)/2)
simp = (ends + 4* odds)* (b - a)/ 6
Do While not cvg And N < maxN
    evens = odds + evens
    oldS = simp  :  odds = 0
    N = 2 * N : h = (b - a) / N
    for k = 1 to N - 1 Step 2
        odds = odds + f(a + k * h)
    Next k
```

h = (b – a)/2
h/3 = (b – a)/6

California State University Northridge 6

Repeated Simpson's Rule 2/2

```
Do While not cvg And N < maxN
    evens = odds + evens
    oldS = simp  :  odds = 0
    N = 2 * N : h = (b - a) / N
    For k = 1 to N - 1 Step 2
        odds = odds + f(a + k * h)
    Next k
    simp = (ends + 4 * odds + _
                2 * evens) * h / 3
    cvg = abs(simp - oldS) <=
            maxRelErr * abs(simp)
Loop
```

Test cvg after end of while loop

Richardson Extrapolation

- Have numerical expression, F, for two different step sizes h and kh
  - Call these F(h) and F(kh)
  - These expressions have a lead error term with order n, $O(h^n)$ (error proportional to h^n)
  - Can get higher order expression by using Richardson Extrapolation formula
    - Typically pick k > 1, but it could be < 1 so long as formula is consistently applied

$$RE = \frac{k^n F(h) - F(kh)}{k^n - 1}$$

Richardson Example

- Use Richardson extrapolation for de^x/dx at x = 1 and h = 0.1 & h = 0.2
  - What are k and n? ***k = h_2/h_1 = 2; n = order = 2***

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2)$$

$$f_i'(h=0.1) = \frac{e^{1.1} - e^{0.9}}{2(0.1)} = 2.722815$$

$$f_i'(h=0.2) = \frac{e^{1.2} - e^{0.8}}{2(0.2)} = 2.736440$$

$$RE = \frac{k^n F(h) - F(kh)}{k^n - 1} = \frac{2^2(2.722815) - 2.736440}{2^2 - 1} = 2.718273$$

- Correct value for de^x/dx at x = 1 is e^1 = 2.7182818… so error is 9.07×10^{-6}

Why so much error reduction?

- Richardson extrapolation removes lead term in infinite series for error
- Look at truncation error series for central-difference, first-derivative

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} \left[\frac{f_i''' h^2}{3!} - \frac{f_i''''' h^4}{5!} - \frac{f_i^{(vii)} h^6}{7!} - \ldots \right]$$

- Richardson extrapolation eliminates the lead error term, $(d^3f/dx^3)_i h^2/3!$, leaving a fourth-order error
  - This is also true in Romberg integration

Romberg Integration

- A technique that starts with a trapezoid rule value for an initial step size, h_{init}
- In a series of iterations, the step size is cut in half for each iteration
- The trapezoid rule is applied to the new step size
- Richardson extrapolation is applied to the trapezoid rule results ***and*** to all previous Richardson extrapolations

Romberg Integration

- Starts with Trapezoid rule result, called T_{00}, using initial step size $(b - a)/N_{init}$ (N_{init} = 1,2)
- Half step size (double N) and get new Trapezoid rule result, called T_{10}
- Use Richardson Extrapolation and double N
  - Extrapolate on T_{00} and T_{10} to get T_{11}
  - Half h again to get new trapezoid rule value T_{20}
  - Extrapolate on T_{10} and T_{20} to get T_{21}
  - Extrapolate on T_{11} and T_{21} to get T_{22}
- For each new N, T_{k0}, extrapolate to get T_{kk}
- Continue until desired error is achieved $|T_{kk} - T_{k,k-1}| <= \text{allowedErr} * |T_{kk}|$

Romberg Integration

- General forms for initial T_{k0} and subsequent T_{km}

$$T_{km} = \frac{4^m T_{k,m-1} - T_{k-1,m-1}}{4^m - 1} \quad m = 1, \cdots, k; \text{ all } k$$

$$T_{k0} = T\left(h_{init}/2^k\right) \quad \text{all } k$$

$$T_{00}$$
$$T_{10} \rightarrow T_{11} = \frac{4^1 T_{10} - T_{00}}{4^1 - 1}$$
$$T_{20} \rightarrow T_{21} = \frac{4^1 T_{20} - T_{10}}{4^1 - 1} \rightarrow T_{22} = \frac{4^2 T_{21} - T_{11}}{4^2 - 1} \qquad h_{init} = \frac{b-a}{2}$$
$$T_{30} \rightarrow T_{31} = \frac{4^1 T_{30} - T_{20}}{4^1 - 1} \rightarrow T_{32} = \frac{4^2 T_{31} - T_{21}}{4^2 - 1} \rightarrow T_{33} = \frac{4^3 T_{32} - T_{22}}{4^3 - 1}$$

Romberg Pseudo Code

```
maxIter = 20; err = 1e-10
N = 2; h = (b - a)/2; cvg = false; k = 0
T00 = h * [f(a) + 2 * f(a+h) + f(b)]/2
While not cvg and k <= maxIter
    k = k + 1; N = 2 * N; h = (b - a)/N
    Make loop to sum f(a+mh) for m = 1 to
                        N - 1 (odd N only)
    Tk0 = Tk-1,0/2 + h * sum
    Make loop for m = 1 to k to get: Tkm =
                (4^m Tk,m-1 - Tk-1,m-1) / (4^m - 1)
    cvg = abs(Tkk - Tk,k-1) <= err * abs(Tkk)
End While
if cvg then ans = Tkk else ans = errorCode
```


Romberg Integration (1-based)

- General forms for initial T_{k1} and subsequent T_{km}

$$T_{km} = \frac{4^{m-1} T_{k,m-1} - T_{k-1,m-1}}{4^{m-1} - 1} \quad m = 2, \cdots, k; \text{ all } k$$

$$T_{k1} = T\left(h_{init}/2^k\right) \quad \text{all } k$$

$$T_{11}$$
$$T_{21} \rightarrow T_{22} = \frac{4^1 T_{21} - T_{11}}{4^1 - 1}$$
$$T_{31} \rightarrow T_{32} = \frac{4^1 T_{31} - T_{21}}{4^1 - 1} \rightarrow T_{33} = \frac{4^2 T_{32} - T_{22}}{4^2 - 1} \qquad h_{init} = \frac{b-a}{2}$$
$$T_{41} \rightarrow T_{42} = \frac{4^1 T_{41} - T_{31}}{4^1 - 1} \rightarrow T_{43} = \frac{4^2 T_{42} - T_{32}}{4^2 - 1} \rightarrow T_{44} = \frac{4^3 T_{43} - T_{33}}{4^3 - 1}$$

Romberg Pseudo Code

```
%For one-based arrays
maxIter = 20; err = 1e-10
N = 2; h = (b - a)/2; cvg = false; k = 1
T11 = h * [f(a) + 2 * f(a+h) + f(b)]/2
While not cvg and k < maxIter
    k = k + 1; N = 2 * N; h = (b - a)/N
    Make loop to sum f(a+mh) for m = 1 to
                        N - 1 (odd N only)
    Tk1 = Tk-1,1/2 + h * sum
    Make loop for m = 2, k to get: Tkm =
          (4^m-1 Tk,m-1 - Tk-1,m-1) / (4^m-1 - 1)
    cvg = abs(Tkk - Tk,k-1) <= err * abs(Tkk)
End While
if cvg then ans = Tkk else ans = errorCode
```


Gauss Quadrature

- Numerical integration algorithm that uses ***fixed x points*** to integrate f(x)
  - High accuracy with small number of points
  - Formula is for integral from –1 to +1

$$\int_{-1}^{1} f(x)dx \approx \sum_{j=1}^{n} \gamma_j f(x_j)$$

  - Choose n and find weights, γ_j, and fixed points for integrand, x_j, in tables
  - Formula with n fixed points, x_j , is exact for a polynomial of order 2n – 1 (or less)

Gauss Quadrature Data

- Fixed (sample) points, x_j, and weights, γ_j, for small values of n typical of those used in finite-element programs

n	x_j	γ_j	n	x_j	γ_j
1	0	2	4	±0.861136312	0.347854845
2	$\pm 1/3^{1/2}$	1		±0.339981044	0.652145155
3	$\pm(0.6)^{1/2}$	5/9	5	±0.906179846	0.236926885
	0	8/9		±0.538469310	0.478628671
				0	0.568888889

x_j values with ± signs used twice (once with + and once with -) with same γ_j

Application to Other Limits

- For integral of f(y) from a to b, use x so that y = a + (b – a)(x + 1)/2
  - dy = (b – a)dx/2
  - At x = –1, y = a
  - At x = 1, y = b

x_j from table

$$y_j = a + \frac{(b-a)(x_j+1)}{2}$$

$$I = \int_{y=a}^{y=b} f(y)dy = \int_{x=-1}^{x=1} f\left[a + \frac{(b-a)(x+1)}{2}\right]\frac{(b-a)}{2}dx =$$

$$\frac{(b-a)}{2}\int_{x=-1}^{x=1} f\left[a + \frac{(b-a)(x+1)}{2}\right]dx \quad I \approx \frac{(b-a)}{2}\sum_{j=1}^{n}\gamma_j f(y_j)$$

Adaptive Quadrature

- Uses uneven values of step size, h
- Makes h smaller in regions where integrand varies sharply
- Algorithms more complex than those considered here (Trapezoid, Simpson, Romberg)
- Several canned routines for numerical quadrature use adaptive quadrature
  - Used in MATLAB function integral (next slide)

MATLAB Function `integral`

- MATLAB function for numerical integration (quadrature) used in assignment 6
  - Use: <result> = integral(<function handle>, <lower limit>, <upper limit>)
    - <function handle> computes integrand, f(x)
    - Must be able to compute array formulas
    - Example: Integrate $e^{-x^2}sin(x)$ from 0 to 1
    - Use function call `I = integral(@myInt, 0, 1)` with function below

```
function f = myInt(x)
    f = exp(-(x.^2)).*sin(x)
```


MATLAB Function `integral` II

- See help for options on `integral`
  - Can use a third argument to function that specified error tolerance instead of the default absolute error of 10^{-6}
  - There are other functions for specialized integration problems
  - Integral can handle infinite limits provided that the integral converges at such limits
- Integral uses recursive adaptive global quadrature

Programming Assignment Six

- See assignment on line
- Write code for Romberg integration in both VBA and MATLAB
  - function I = romberg(f, a, b) *MATLAB*
  - Function Romberg(f As String, a As _ Double, b as Double) As Double *VBA*
- MATLAB code has function handle for integrand as input to function
- VBA code uses Application.Run method for integrand