Chapter 4: Numerical Integration

FuSen Lin

Department of Computer Science and Engineering
National Taiwan Ocean University

Scientific Computing, Fall 2010

4 Numerical Integration

Introduction to Numerical Integration

- An m-point Quadrature rule Q for the definite integral

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

  is an approximation of the form

$$Q = (b-a)\sum_{k=1}^{m} \widetilde{w}_k f(x_k) = \sum_{k=1}^{m} w_k f(x_k).$$

- The x_k are the **abscissas** and the w_k are the **weights**. The abscissas and weights define the rule, called as **quadrature rule**, and are chosen so that $Q \approx I$.
- **Efficiency** *essentially depends upon* the **number of function evaluations**.

Introduction to Numerical Integration (2)

- Because the time needed to evaluate f at the x_i is typically much greater than the time needed to form the required linear combination of function values.
- For instance, a six-point quadrature rule is twice as expensive as a three-point rule.
- The quadrature rule can basically be classified into two families: the **Newton-Cotes rules** and **Gauss quadrature rules**.

The Newton-Cotes Rules

- The Newton-Cotes family of quadrature rules are derived by integrating *uniformly spaced polynomial interpolants* of the integrand. This means that to find a polynomial approximation $p(x)$ of the integrand $f(x)$ and integrate $p(x)$ so that

$$\int_a^b f(x)\,dx \approx \int_a^b p(x)\,dx$$

- The m-point Newton-Cotes rule is defined by

$$Q_{NC(m)} = \int_a^b p_{m-1}(x)\,dx = h\sum_{k=1}^{m} c_k f(x_k)$$

where $p_{m-1}(x)$ interpolates $f(x)$ at

$$x_k = a + (k-1)h, \quad h = \frac{b-a}{m-1}, \quad k = 1:m.$$

The Trapezoidal Rule

- If $m = 2$, then

$$f(x) \approx p_1(x) = f(a) + \frac{f(b) - f(a)}{b - a}(x - a),$$

and thus we obtain the **trapezoidal rule**:

$$\begin{aligned} Q_{NC}(2) &= \int_a^b p_1(x)dx = \int_a^b \left(f(a) + \frac{f(b) - f(a)}{b - a}(x - a) \right) dx \\ &= (b - a)\left(\frac{1}{2}f(a) + \frac{1}{2}f(b) \right). \end{aligned}$$

- In this rule the weights are $\widetilde{w}_1 = \widetilde{w}_2 = 1/2$.

The Simpson Rule

- If $m = 3$ and $c = (a+b)/2$, then

$$f(x) \approx p_2(x) = \alpha x^2 + \beta x + \gamma \quad \text{or}$$

$$p_2(x) = f(a) + \frac{f(c)-f(a)}{c-a}(x-a) + \frac{\frac{f(b)-f(c)}{b-c} - \frac{f(c)-f(a)}{c-a}}{b-a}(x-a)(x-c)$$

and thus we obtain the Simpson rule:

$$Q_{NC}(3) := \frac{b-a}{3}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

- In this rule the weights are $\widetilde{w}_1 = 1/3$, $\widetilde{w}_2 = 4/3$, and $\widetilde{w}_3 = 1/3$.

General Newton-Cotes Rule (1)

- For general m, we apply the Newton form of interpolating polynomial

$$p_{m-1}(x) = \sum_{k=1}^{m} \left(c_k \prod_{i=1}^{k-1} (x - x_i) \right)$$

to approximate $f(x)$ and obtain the m-point Newton-Cotes Rule

$$Q_{NC}(m) := \int_a^b p_{m-1}(x)dx = \sum_{k=1}^{m} c_k \int_a^b \left(\prod_{i=1}^{k-1} (x - x_i) \right) dx.$$

General Newton-Cotes Rule (2)

- If we set $x = a + sh$ (s be integer) then

$$Q_{NC}(m) := \int_a^b p_{m-1}(x)dx = h\int_0^{m-1} p_{m-1}(a+sh)ds = \sum_{k=1}^{m} c_k h^k S_{mk},$$

where

$$S_{mk} = \int_0^{m-1}\left(\prod_{i=1}^{k-1}(s-i+1)\right)ds.$$

General Newton-Cotes Rule (3)

- The c_k are **divided differences**. Because of the equal spacing, the divided differences c_k have a simple form in terms of $f_i = f(x_i)$, as was shown in Sec.2.4.1 (p.95). For example,

$$
\begin{aligned}
c_1 &= f_1 \\
c_2 &= (f_2 - f_1)/h \\
c_3 &= (f_3 - 2f_2 + f_1)/(2h^2) \\
c_4 &= (f_4 - 3f_3 + 3f_2 - f_1)/(3!h^3)
\end{aligned}
$$

General Newton-Cotes Rule (4)

- Recipes for the S_{mk} can also be derived. Here are a few examples:

$$\begin{aligned}
S_{m1} &= \int_0^{m-1} 1\,ds = (m-1) \\
S_{m2} &= \int_0^{m-1} s\,ds = (m-1)^2/2 \\
S_{m3} &= \int_0^{m-1} s(s-1)\,ds = (m-1)^2(m-5/2)/3 \\
S_{m4} &= \int_0^{m-1} s(s-1)(s-2)\,ds = (m-1)^2(m-3)^2/4
\end{aligned}$$

General Newton-Cotes Rule(5)

- Using these tabulations we can readily derive the weights for any particular m-point Rule. For example, if $m = 4$ then $S_{41} = 3$, $S_{42} = 9/2$, $S_{43} = 9/2$, and $S_{44} = 9/4$. Thus,

$$\begin{aligned} Q_{NC}(4) &= S_{41}c_1h + S_{42}c_2h^2 + S_{43}c_3h^3 + S_{44}c_4h^4 \\ &= \frac{3h}{8}(f_1 + 3f_2 + 3f_3 + f_4) \\ &= (b-a)(f_1 + 3f_2 + 3f_3 + f_4)/8 \end{aligned}$$

- The weight vector for $Q_{NC}(4)$ is $[1, 3, 3, 1]/8$.

Implementation of Newton-Cotes Rule

- For convenience in subsequent computations, we package the weight vectors of the Newton-Cotes Rules in the *function* **NCWeights.m**.
- Notice that the weight vectors are symmetric about their middle in that $w(1:m) = w(m:-1:1)$.
- The evaluation of $Q_{NC(m)}$ is a scaled inner product of the weight vector w and the vector of function values:

$$Q_{NC(m)} = (b-a)\sum_{k=1}^{m} w_k f_k = (b-a)[w_1 \cdots w_m] \begin{bmatrix} f(x_1) \\ \vdots \\ \vdots \\ f(x_m) \end{bmatrix}$$

- We have the *function* **QNC.m** for $2 <= m <= 11$.

Error of Newton-Cotes Rules

- The errors of Newton-Cotes rules depend on the *quality of polynomial interpolant*. Here is an **error bound of Simpson's rule**:
- **Theorem 4:** If $f(x)$ and its *first four derivatives are continuous* on $[a, b]$, then

$$\left| \int_a^b f(x)dx - Q_{NC(3)} \right| \leq \frac{(b-a)^5}{2880} M_4$$

where M_4 is an upper bound of $|f^{(4)}(x)|$ on $[a, b]$.
- PROOF: Suppose

$$p(x) = c_1 + c_2(x-a) + c_3(x-a)(x-b) + c_4(x-a)(x-b)(x-c)$$

is the Newton form of the cubic interpolant to $f(x)$ at the points a, b, c, and d.

Proof of Theorem 4 (1)

- If c is the midpoint of the interval $[a, b]$, then

$$\int_a^b (c_1 + c_2(x-a) + c_3(x-a)(x-b))dx = Q_{NC(3)},$$

because the first three terms in the expression for $p(x)$ specify the quadratic interpolant of $(a, f(a))$, $(c, f(c))$, $(b, f(b))$, on which the three-point Newton-Cotes rule is based.

- By symmetry we have

$$\int_a^b (x-a)(x-b)(x-c)dx = 0$$

and so

$$\int_a^b p(x)dx = Q_{NC(3)}$$

Proof of Theorem 4 (2)

- The error in $p(x)$ is given by Theorem 2 (p.90),

$$f(x) - p(x) = \frac{f^{(4)}(\eta_x)}{24}(x-a)(x-b)(x-c)(x-d)$$

and thus,

$$\int_a^b f(x)dx - Q_{NC(3)} = \int_a^b \left(\frac{f^{(4)}(\eta_x)}{24}(x-a)(x-b)(x-c)(x-d) \right) dx.$$

- Taking absolute values, we obtain

$$\left| \int_a^b f(x)dx - Q_{NC(3)} \right| \leq \frac{M_4}{24} \int_a^b |(x-a)(x-b)(x-c)(x-d)| dx.$$

Proof of Theorem 4 (3)

- If we set $d = c$, then $(x-a)(x-b)(x-c)(x-d)$ is always negative and it is easy to verify that

$$\int_a^b |(x-a)(x-b)(x-c)(x-d)|dx = \frac{(b-a)^5}{120}$$

and so

$$\left|\int_a^b f(x)dx - Q_{NC(3)}\right| \leq \frac{M_4}{24}\frac{(b-a)^5}{120} = \frac{M_4}{2880}(b-a)^5.$$

Error of Newton-Cotes Rules

- For another proof, please see Numerical Analysis.
- Note that if $f(x)$ is a *cubic* polynomial then $f^{(4)}(x) = 0$ and so Simpson's rule is *exact*. This is somewhat surprising because the rule is based on the integration of a *quadratic* interpolant.

Error Formula of Newton-Cotes Rule

- In general, it can be shown that

$$\int_a^b f(x)dx = Q_{NC(m)} + c_m f^{(d+1)}(\eta)\left(\frac{b-a}{m-1}\right)^{d+2}$$

where c_m is a small constant, $\eta \in [a,b]$, and

$$d = \begin{cases} m-1, & \text{if } m \text{ is even,} \\ m, & \text{if } m \text{ is odd.} \end{cases}$$

- Notice that if m is *odd*, such as Simpson's rule, then an *extra degree of accuracy* results.
- If $|f^{(d+1)}(x)| \leq M_{d+1}$ on $[a,b]$, then

$$\left|\int_a^b f(x)dx - Q_{NC(m)}\right| \leq |c_m| M_{d+1} h^{d+2}, \quad h = \frac{b-a}{m-1}$$

Open Newton-Cotes Rules

- The Newton-Cotes rules presented previously are actually the *closed* Newton-Cotes rules because $f(x)$ is evaluated at the *left and right endpoints*.
- The m-point *open* Newton-Cotes rule places the abscissas at $a + ih$ where $h = (b - a)/(m + 1)$ and $i = 1 : m$.
- The one-point open Newton-Cotes rule is called the **midpoint rule**.
- Note that for $m = 3, 5, 6, 7, ...$, the open rules involve *negative* weights, a feature that can *undermine the numerical stability* of the rule.
- The closed rules do not *go negative* weights until $m = 9$, making them a more attractive family of quadrature rules from this point of view. However, the open rules can be useful when f has *endpoint singularities*.

Composite Rules (1)

- The **Composite Newton-Cotes rules** are to *partition the interval* $[a, b]$ *into n subintervals* which are sufficiently small, and then *apply* $Q_{NC(m)}$ *to each subinterval*.
- That means, if we have a partition

$$a = z_1 < z_2 < \cdots < z_{n+1} = b,$$

then

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} \int_{z_i}^{z_{i+1}} f(x)\,dx \approx \sum_{i=1}^{n} Q_{NC(m)}^{(i)}$$

which is the **composite quadrature rule** based on Newton-Cotes Results.

Composite Rules (2)

- For example, let $\Delta_i = z_{i+1} - z_i$ and $z_{i+1/2} = (z_i + z_{i+1})/2$ for $i = 1 : n$, if we apply the Simpson rule to each subinterval $[z_i, z_{i+1}]$ then we have a **composite Simpson rule**

$$Q_{\text{Simp}} = \sum_{i=1}^{n} \frac{\Delta_i}{6}(f(z_i) + 4f(z_{i+1/2}) + f(z_{i+1})).$$

- In general, if z houses a partition $[a, b]$ and **fname** is a string that names a function, then

$$\text{numI} = 0;$$
$$\textbf{for } i = 1 : \text{length}(z) - 1,$$
$$\quad \text{numI} = \text{numI} + \text{QNC}('\text{fname}', z(i), z(i+1), m);$$
$$\textbf{end}$$

assigns to **numI** the the composite m-point Newton-Cotes estimate of the integral based on the partition housed in z.

Composite Rules (3)

- We next focus on composite rules that are based on *uniformly partitions.* In these rules, we have

$$z_i = a + (i-1)\Delta, \quad \Delta = \frac{b-a}{n}, \quad i = 1 : n+1.$$

- Thus the composite rule evaluation has the form:

$$\begin{aligned}
&\mathrm{numI} = 0;\\
&\mathrm{Delta} = (b-a)/n;\\
&z = a + [0:n] * \mathrm{Delta};\\
&\textbf{for } i = 1:n,\\
&\quad \mathrm{numI} = \mathrm{numI} + \mathrm{QNC}('\mathrm{fname}', z(i), z(i+1), m);\\
&\textbf{end}
\end{aligned}$$

- This is the composite m-point Newton-Cotes rule with an n-subinterval partition, denoted as $Q_{NC(m)}^{(n)}$.

Composite Rules (4)

- However, the computation is a little *inefficient* since it involves $n - 1$ *extra function evaluations* and a **for-loop**. The rightmost f-evaluation in the ith call to **QNC** is the same as the leftmost f-evaluation in the (i+1)st call.
- To avoid redundant f-evaluation and a **for**-loop with repeated function calls, it is better not to apply **QNC** to each n subintervals.
- Instead, we pre-compute all the required function evaluations and store them in a single vector fval$(1 : n * (m - 1) + 1)$. The linear combination that defines the composite rule is then calculated.

Composite Rules (5)

- In the preceding $Q^{(4)}_{NC(5)}$ example, the 17 required function evaluations are assembled in fval$(1:17)$ If w is the weight vector for $Q_{NC(5)}$, then

$$Q^{(4)}_{NC(5)} = \Delta(w^T\text{fval}(1:5) + w^T\text{fval}(5:9) + w^T\text{fval}(9:13) + w^T\text{fval}(13:17)).$$

- This concludes that $Q^{(n)}_{NC(m)}$ is a *summation of n inner products*, each of which involves the weight vector w of the underlying rule and a portion of the *fval*-vector (see **compQNC.m**).

Error of the Composite Newton-Cotes Rules (1)

- Suppose Q_i is the m-point Newton-Cotes estimate of the ith subinterval. If this rule is exact for polynomial of degree d, then using Eq.(4.2) we obtain

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} \int_{z_i}^{z_{i+1}} f(x)\,dx = \sum_{i=1}^{n} \left(Q_i + c_m f^{(d+1)}(\eta_i) \left(\frac{z_{i+1} - z_i}{m-1} \right)^{d+2} \right).$$

- By definition

$$Q_{NC(m)}^{(n)} = \sum_{i=1}^{n} Q_i \quad and \quad z_{i+1} - z_i = \Delta = \frac{b-a}{n}.$$

Moreover, it can be shown that

$$\frac{1}{n} \sum_{i=1}^{n} f^{(d+1)}(\eta_i) = f^{(d+1)}(\eta) \quad \text{for some} \quad \eta \in [a, b].$$

Error of the Composite Newton-Cotes Rules (2)

- We hence have

$$\int_a^b f(x)\,dx = Q_{NC(m)}^{(n)} + c_m \left(\frac{b-a}{n(m-1)}\right)^{d+2} n f^{(d+1)}(\eta).$$

If $|f^{(d+1)}(x)| \le M_{d+1}$ for all $x \in [a,b]$, then

$$\left| Q_{NC(m)}^{(n)} - \int_a^b f(x)\,dx \right| \le \left[|c_m| M_{d+1} \left(\frac{b-a}{m-1}\right)^{d+2} \right] \frac{1}{n^{d+1}}.$$

- Comparing with (4.3), we see that the error in **composite rule** is the error in corresponding *simple rule* divided by n^{d+1}. Then, with m fixed it is possible to exercise error control by choosing n sufficiently large.

Error of the Composite Simpson Rule

- For example, suppose we want to approximate the integral with a uniformly spaced composite Simpson rule so that the error is less than a prescribed tolerance *tol*. If we know that $|f^{(4)}(x)| \leq M_4$, then we choose n so that

$$\frac{1}{90} M_4 \left(\frac{b-a}{2}\right)^5 \frac{1}{n^4} \leq tol.$$

- To keep the number of f-evaluations as small as possible, n should be the *smallest positive integer* that satisfies

$$n \geq (b-a) \sqrt[4]{\frac{M_4(b-a)}{2880 \cdot tol}}$$

- Exercising the script file **ShowCompQNC.m**, you will see the error properties of the composite Newton-Cotes rules.

Adaptive Quadrature Methods

- Uniformly spaced composite rules that are exactly for degree d polynomials are efficient if $f^{(d+1)}$ is uniformly behaved across $[a, b]$. However, if the magnitude of $f^{(d+1)}$ varies widely across the interval of integration, then the error control process discussed in Sec. 4.2 may result in an unnecessary number of function evaluations.
- This is because n is determined by an interval-wide derivative bound M_{d+1}. In regions where $f^{(d+1)}$ is small compared to this value, the subintervals are (possibly) much shorter than necessary.
- Adaptive quadrature methods can resolve this problem by 'discovering' where the integrand is ill-behaved and shortening the subintervals accordingly.

An Adaptive Newton-Cotes Procedure (1)

- The **adaptive Newton-Cotes procedure** is *similar to* the one we developed for **adaptive piecewise linear interpolation**. In order to obtain a good partition of $[a, b]$, we need to be able to estimate error. If the error is not small enough, then the partition should be refined.
- We first fix m (the same point rule) and develop a method for estimating the error: Let $A_1 = Q^{(1)}_{NC(m)}$ and $A_2 = Q^{(2)}_{NC(m)}$, where A_1 is the simple m-point rule estimate and A_2 is the two-subinterval, m-point rule estimate.

An Adaptive Newton-Cotes Procedure (2)

- If these rules are exact for degree d polynomials, then it can be shown that

$$I = A_1 + \left[c_m f^{(d+1)}(\eta_1) \left(\frac{b-a}{m-1} \right)^{d+2} \right]$$

$$I = A_2 + \left[c_m f^{(d+1)}(\eta_2) \left(\frac{b-a}{m-1} \right)^{d+2} \right] \frac{1}{2^{d+1}}$$

where η_1 and η_2 are in the interval $[a, b]$.

- We now assume that $f^{(d+1)}(\eta_1) = f^{(d+1)}(\eta_2)$, which is reasonable if $f^{(d+1)}$ does not vary much on $[a, b]$. Thus, it can be written as

$$I = A_1 + C, \quad \text{and} \quad I = A_2 + \frac{C}{2^{d+1}}, \quad C = \left[c_m f^{(d+1)}(\eta_1) \left(\frac{b-a}{m-1} \right)^{d+2} \right].$$

An Adaptive Newton-Cotes Procedure (3)

- By subtracting these two equations for I from each other and solving for C, we obtain

$$C = \frac{A_2 - A_1}{1 - 1/2^{d+1}} \quad \text{and} \quad |I - A_2| \approx \frac{|A_2 - A_1|}{2^{d+1} - 1}.$$

- Thus, this discrepancy provides a reasonable estimate of the error in A_2. If our goal is to proximate I that has absolute error *tol* or less, then the recursive procedure may be organized as **AdaptQNC.m**.

An Adaptive Newton-Cotes Procedure (4)

- If the heuristic estimate of the error is greater than *tol*, then two recursive call are initiated to obtain the approximations

$$Q_L \approx \int_a^{mid} f(x)dx = I_L \quad \text{and} \quad Q_R \approx \int_{mid}^{b} f(x)dx = I_L$$

that satisfy

$$|I_L - Q_L| \leq \frac{tol}{2} \quad \text{and} \quad |I_R - Q_R| \leq \frac{tol}{2}$$

- Setting $Q = Q_L + Q_R$, we see that

$$|I - Q| = |(I_L - Q_L) + (I_R - Q_R)| \leq |I_L - Q_L| + |I_R - Q_R| \leq \frac{tol}{2} + \frac{tol}{2} = tol.$$

- The script file **ShowAdapts.m** illustrates the behavior of **AdaptQNC.m** for various of tolerance and m.

An Adaptive Newton-Cotes Procedure (5)

- Notes: the script **ShowAdapts** uses MATLAB's *global variable capability* in order to report on the *number of function evaluations* that are required for each **AdaptQNC** call.
- The command

  global FunEvals VecFunEvals

  designates **FunEvals** and **VecFunEvals** as *global variables*. They 'sit' in the MATLAB *workspace* and are *accessible by any function* that also designates the two variables as *global*.

MATLAB's Numerical Integrators (1)

- MATLAB has two adaptive quadrature procedures, **quad.m** and **quad8.m**. The first is based on $Q_{NC(3)}$ (Simpson's rule) and the second on $Q_{NC(9)}$.
- We look at **quad.m**. As for **quad8.m**, the calling sequence is identical. A call of the form

$$Q = \text{quad}(\text{'f'}, a, b)$$

  assigns to Q an estimate of the integral of $f(x)$ from a to b. The default relative error tolerance is 10^{-6} (MATLAB 6).
- Otherwise, a fourth input parameter can be used to specify the required tolerance. For example,

$$Q = \text{quad}(\text{'f'}, a, b, tol)$$

MATLAB's Numerical Integrators (2)

- A fifth nonzero parameter can be used to produce a plot of f that reveals where it is evaluated by quad:

$$Q = \text{quad}(\text{'f'}, a, b, tol, 1)$$

- The **number of function evaluations** can be obtained by specifying a *second output parameter*:

$$[Q, \text{count}] = \text{quad}(\text{'f'}, a, b, tol, 1)$$

- In the script file **ShowQuads.m**, these two procedures are applied to the integral of the built-in MATLAB function **humps** that implements

$$\text{humps}(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6$$

on $[0, 1]$.

MATLAB's Numerical Integrators (3)

- It is sometimes the case that the integrand *depends on one or more parameters*. For example, suppose we want to compute

$$G(\alpha, \beta) = \int_0^1 e^{\alpha x} \sin(\beta \pi x) dx$$

  for various α and β.

- In this case we would start by writing a function that *includes the parameters as arguments*, e.g.,

$$\text{function } \; y = \text{F4-3-2}(x, \alpha, \beta)$$

$$y = \exp(\alpha * x). * \sin(\beta * \pi * x);$$

Special Quadratures

- We here discuss two other approaches to the quadrature problem. **Gauss quadrature** is useful in certain specialized settings, as *when there are endpoint singularities*.
- In situations where the *functions evaluations are experimentally determined*, **spline quadrature** has a certain appeal.

Gauss Quadrature

- In the **Newton-Cotes** framework, the integrand is sampled at *regular intervals across* $[a, b]$. For m-point rule, the Newton-Cotes method is *exact* for the degree m (or $m + 1$) polynomials.
- In the **Gauss quadrature**, the **abscissas** are positioned in such a way so that the rule is *correct (exact) for polynomials of maximal degree* (up to $2m - 1$).
- This means, Gauss quadrature chooses the *optimal points* for f-evaluation, rather than *equally spaced* way.

Gauss Quadrature (Continuing)

- The nodes $x_1, x_2, \ldots, x_m \in [a, b]$, and weights $w_1, w_2, \ldots, w_m$ are chosen to *minimize the expected error* in performing the approximation

$$\int_a^b f(x)\,dx \approx \sum_{k=1}^{m} w_k f(x_k)$$

  for an arbitrary integrable function f.
- The simplest Gaussian rule is the **two-point rule**: Let us to determine the abscissas x_1, x_2 and weights w_1, w_2 so that

$$w_1 f(x_1) + w_2 f(x_1) = \int_{-1}^{1} f(x)\,dx$$

  for polynomials of degree 3 $(= 2(2) - 1)$ or less.

Gauss Quadrature (Continuing)

- This means that the rule must be exact for the functions $1, x, x^2$, and x^3 and we obtain the equations

$$\begin{aligned} w_1 + w_2 &= 2 \\ w_1 x_1 + w_2 x_2 &= 0 \\ w_1 x_1^2 + w_2 x_2^2 &= 2/3 \\ w_1 x_1^3 + w_2 x_2^3 &= 0. \end{aligned}$$

- By solving the system of 4 equations, we get $x_2 = -x_1 = 1/\sqrt{3}$ and $w_1 = w_2 = 1$. Thus, for any function $f(x)$ we have

$$\int_{-1}^{1} f(x)\,dx = f(-1/\sqrt{3}) + f(1/\sqrt{3}).$$

This is the two-point Gauss-Legendre rule.

Gauss-Legendre Quadrature

- The m-point **Gauss-Legendre** rule has the form

$$Q_{\mathrm{GL}(m)} = w_1 f(x_1) + \cdots + w_m f(x_m),$$

  where the x_i and w_i are chosen to make the rule *exact* for polynomials of degree $2m-1$.
- One way to *determine the m nodes* (x_i) *and m weights* (w_i) is by solving the $2m$ nonlinear equations

$$w_1 x_1^k + \cdots + w_m x_m^K = \frac{1-(-1)^{k+1}}{k+1}, \quad k = 0 : 2m-1.$$

  The kth equation is obtained by the requirement that the rule

$$w_1 f(x_1) + \cdots + w_m f(x_m) = \int_{-1}^{1} f(x)\,dx$$

  be *exact* for $f(x) = x^k$. It has a *unique* solution.

Gauss-Legendre Quadrature (Continuing)

- This technique could be used to *determine the nodes and weights* for any m-point formula that give *exact* results for polynomials of degree $2m-1$ or less. However, an alternative method can be used to obtain the nodes and weights more easily by applying the *roots* of **orthogonal polynomials**.
- A family of orthogonal polynomials $\{p_0(x), p_1(x), ..., p_n(x), ...\}$, defined on $[a, b]$, has the property (inner product) that

$$< p_i(x), p_j(x) >= \int_a^b p_i(x)\, p_j(x)\, dx = \begin{cases} 0, & \text{if } i \neq j, \\ C_i, & \text{if } i = j, \text{ where } C_i \text{ constant.} \end{cases}$$

Gauss-Legendre Quadrature (Continuing)

- For example, one of the most famous *orthogonal polynomial families* is the **Legendre polynomials** defined on $[-1, 1]$, the first few Legendre polynomials are

$$p_0(x) = 1, \quad p_1(x) = x, \quad p_2(x) = x^2 - \frac{1}{3},$$

$$p_3(x) = x^3 - \frac{3}{5}x, \quad p_4(x) = x^4 - \frac{6}{7}x^2 + \frac{3}{35}, \cdots, \text{etc.}$$

- The rule using the *roots of Legendre polynomials* as notes and their corresponding weights is called **Gauss-Legendre quadrature**, which is defined as the following theorem.

Gauss-Legendre Theorem

- **Theorem:** Suppose that $x_1, x_2, \ldots, x_m$ are the *roots* of the n**th Legendre polynomials** $p_n(x)$ and that for each $i = 1 : m$, the weights w_i are defined by

$$w_i = \int_{-1}^{1} \prod_{\substack{j=1 \\ j \neq i}}^{m} \frac{x - x_j}{x_i - x_j} dx.$$

If $f(x)$ is any polynomial of degree less than $2m$, then

$$\int_{-1}^{1} f(x)dx = \sum_{i=1}^{m} w_i f(x_i)$$

- For example, the **two-point rule** is using the roots of $p_2(x) = x^2 - \frac{1}{3}$ $(\pm 1/\sqrt{3})$ and their corresponding weights $w_1 = w_2 = 1$.

Gauss-Legendre Quadrature (Continuing)

- The **three-point rule** is using the roots of $p_3(x) = x^3 - \frac{3}{5}x$ (0 and $\pm\sqrt{3/5}$) and their corresponding weights $w_1 = w_3 = 5/9$ and $w_2 = 8/9$. That is,

$$\int_{-1}^{1} f(x)\,dx \approx \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right).$$

- The rule, defined on $[-1, 1]$,

$$Q_{\mathrm{GL}(m)} = w_1 f(x_1) + \cdots + w_m f(x_m) \approx \int_{-1}^{1} f(x)\,dx$$

can easily be *transformed into any definite integrals on* $[a, b]$, by a *change of variable*:

$$\int_a^b f(x)\,dx = \frac{b-a}{2} \int_{-1}^{1} g(x)\,dx, \quad g(x) = f\left(\frac{a+b}{2} + \frac{b-a}{2}x\right).$$

Error of Gauss-Legendre Quadrature

- It can be shown that

$$\left| \int_a^b f(x)\,dx - Q_{\mathrm{GL}(m)} \right| \le \frac{(b-a)^{2m+1}(m!)^4}{(2m+1)[(2m)!]^3} M_{2m},$$

  where M_{2m} is a constant that bounds $|f^{(2m)}|$ on $[a, b]$.
- The script file **GLvsNC.m** compares the $Q_{\mathrm{GL}(m)}$ and $Q_{\mathrm{NC}(m)}$ rules when they are applied to the integral of $\sin(x)$ from 0 to $\pi/2$.

Spline Quadrature

- Suppose $S(x)$ is a **cubic spline interpolant** of (x_i, y_i), $i = 1 : n$ and that we wish to compute $I = \int_{x_1}^{x_n} S(x)\,dx$. If the *ith local cubic* is represented by

$$q_i(x) = \rho_{i,4} + \rho_{i,3}(x - x_i) + \rho_{i,2}(x - x_i)^2 + \rho_{i,1}(x - x_i)^3,$$

then

$$\int_{x_i}^{x_{i+1}} q_i(x)\,dx = \rho_{i,4}h_i + \frac{\rho_{i,3}}{2}h_i^2 + \frac{\rho_{i,2}}{3}h_i^3 + \frac{\rho_{i,1}}{4}h_i^4$$

where $h_i = x_{i+1} - x_i$.

- By *summing these quantities* from $i = 1 : n - 1$, we obtain the sought after spline integral. The *function* **SplineQ.m** in MATLAB can be used for **spline quadrature**.
- The script file **ShowSplineQ.m** uses this function to produce the estimates for the integral of $\sin(x)$ from 0 to $\pi/2$.