From `quadl` to `quadgk`

L.F. Shampine  
Professor Emeritus  
Southern Methodist University  
lfshampine@aol.com  
faculty.smu.edu/shampine/

This talk is based on work done with  
M.E. Hosea  
The MathWorks, Inc.
Quadrature Formulas

\[ I(f) = \int_{a}^{b} f(x) \, dx \approx Q(f) = \sum_{j=1}^{M} w_j f(x_j) \]

All popular formulas have weights \( w_j > 0 \) for all \( j \) and nodes \( x_j \) that satisfy

\[ a \leq x_1 < x_2 < \ldots < x_{M-1} < x_M \leq b \]
Quadrature Formulas

\[ I(f) = \int_a^b f(x) \, dx \approx Q(f) = \sum_{j=1}^{M} w_j f(x_j) \]

All popular formulas have weights \( w_j > 0 \) for all \( j \) and nodes \( x_j \) that satisfy

\[ a \leq x_1 < x_2 < \ldots < x_{M-1} < x_M \leq b \]

The error is a linear functional

\[ E(f) = I(f) - Q(f) \]
If $Q^*(f)$ is more accurate than $Q(f)$,

$$R(f) = Q^*(f) - Q(f) = E(f) - E^*(f) \approx E(f)$$
If $Q^*(f)$ is more accurate than $Q(f)$,

$$R(f) = Q^*(f) - Q(f) = E(f) - E^*(f) \approx E(f)$$

Evaluating $f(x)$ is the main cost. To reduce this cost, use a $Q^*(f)$ that shares many nodes with $Q(f)$. 
If $Q^*(f)$ is more accurate than $Q(f)$,

$$R(f) = Q^*(f) - Q(f) = E(f) - E^*(f) \approx E(f)$$

Evaluating $f(x)$ is the main cost. To reduce this cost, use a $Q^*(f)$ that shares many nodes with $Q(f)$.

A formula has degree of precision $d$ if it integrates exactly all polynomials of degree $d$. 
Pairs in MATLAB Quadrature Programs

<table>
<thead>
<tr>
<th></th>
<th>$d$</th>
<th>$d^*$</th>
<th># nodes</th>
<th>resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>quad</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>25 %</td>
</tr>
<tr>
<td>quadl</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>23 %</td>
</tr>
<tr>
<td>quadgk</td>
<td>13</td>
<td>23</td>
<td>15</td>
<td>10 %</td>
</tr>
</tbody>
</table>

The resolution of a pair (in percent) is

$$100 \times \left( \frac{1}{b-a} \right) \times \max_j (x_{j+1} - x_j)$$
Using linearity,

\[ \int_{a}^{b} f(x) \, dx = \int_{\alpha_1}^{\beta_1} f(x) \, dx + \ldots + \int_{\alpha_N}^{\beta_N} f(x) \, dx \]

with \( a = \alpha_1 < \beta_1 < \ldots < \beta_{N-1} = \alpha_N < \beta_N = b \).

Let \( h = \max (\beta_j - \alpha_j) \).
Using linearity,
\[ \int_a^b f(x) \, dx = \int_{\alpha_1}^{\beta_1} f(x) \, dx + \ldots + \int_{\alpha_N}^{\beta_N} f(x) \, dx \]

with \( a = \alpha_1 < \beta_1 < \ldots < \beta_{N-1} = \alpha_N < \beta_N = b \).
Let \( h = \max (\beta_j - \alpha_j) \).

\[ Q_j(f) \approx I_j(f) = \int_{\alpha_j}^{\beta_j} f(x) \, dx \]
\[ Q_j^*(f) - Q_j(f) = R_j(f) \approx E_j(f) \]
\[ \sum_{j=1}^N Q_j(f) \approx I(f), \quad \sum_{j=1}^N R_j(f) \approx E(f) \]
Adaptive Quadrature

To get an approximation $Q(f)$ with error smaller than a tolerance $\tau$, codes based on compound formulas refine the mesh. **Adaptive** codes split only the subintervals where the integrand is relatively difficult to approximate.
Adaptive Quadrature

To get an approximation $Q(f)$ with error smaller than a tolerance $\tau$, codes based on compound formulas refine the mesh. *Adaptive* codes split only the subintervals where the integrand is relatively difficult to approximate.

- Some codes split the subinterval with the largest error. Minimizes evaluations of $f(x)$, but the overhead is high.
Adaptive Quadrature

To get an approximation \( Q(f) \) with error smaller than a tolerance \( \tau \), codes based on compound formulas refine the mesh. **Adaptive** codes split only the subintervals where the integrand is relatively difficult to approximate.

- Some codes split the subinterval with the largest error. Minimizes evaluations of \( f(x) \), but the overhead is high.

- `quadgk` *halves* \((\alpha_j, \beta_j)\) if \(|R_j(f)| > \tau (\beta_j - \alpha_j)/(b - a)\). This test is **local** and **conservative**:

\[
|E(f)| \leq \sum_{j=1}^{N} |E_j(f)| \approx \sum_{j=1}^{N} |R_j(f)| \leq \sum_{j=1}^{N} \tau \frac{\beta_j - \alpha_j}{b - a} = \tau
\]
Vectorization

The MATLAB quadrature programs require that the function for evaluating the integrand accept a vector $x$ and return the vector $f(x)$ because the cost depends weakly on the length of $x$ when fast built-in functions and array operations are used. They form all the $f(x_j)$ of $Q(f)$ and $Q^*(f)$ in a single call. Accordingly, the cost of evaluating the pair is about the same in all the programs.
Vectorization

The MATLAB quadrature programs require that the function for evaluating the integrand accept a vector \( x \) and return the vector \( f(x) \) because the cost depends weakly on the length of \( x \) when fast built-in functions and array operations are used. They form all the \( f(x_j) \) of \( Q(f) \) and \( Q^*(f) \) in a single call. Accordingly, the cost of evaluating the pair is about the same in all the programs.

`quad` and `quadl` process subintervals one at a time, but `quadgk` processes them simultaneously. In this, `quadgk` does all the function evaluations for all subintervals in a single call. This is very advantageous in MATLAB.
Resolution

The solvers start with an *a priori* split of \([a, b]\).

- **quad** splits \([a, b]\) into three unequal subintervals. Minimum of 13 samples in 4 function calls.
- The initialization of **quadl** is more complicated. Minimum of 18 samples in 2 function calls.
- **quadgk** splits \([a, b]\) into 10 equal subintervals. Minimum of 150 samples in one function call.

**quadgk** has a much better initial resolution of the behavior of the integrand and so is much less likely to be deceived.
Usual Error Representation

If $f \in C^{d+1}$, popular formulas applied to the interval $[a, b]$ satisfy

$$E(f) = c_Q \left( \frac{b-a}{2} \right)^{d+2} f^{(d+1)}(\xi)$$
Usual Error Representation

If \( f \in C^{d+1} \), popular formulas applied to the interval \([a, b]\) satisfy

\[
E(f) = c_Q \left( \frac{b - a}{2} \right)^{d+2} f^{(d+1)}(\xi)
\]

- Splitting \((\alpha_j, \beta_j)\) reduces \(E_j(f)\) and the bigger \(d\), the faster the error is reduced.
Usual Error Representation

If \( f \in C^{d+1} \), popular formulas applied to the interval \([a, b]\) satisfy

\[
E(f) = cQ\left(\frac{b - a}{2}\right)^{d+2} f^{(d+1)}(\xi)
\]

- Splitting \((\alpha_j, \beta_j)\) reduces \( E_j(f) \) and the bigger \( d \), the faster the error is reduced.

- If \( f \in C^{d^*+1} \), the error estimate is asymptotically correct:

\[
Q^*(f) - Q(f) = E(f) - E^*(f) = E(f) + O(h^{d^*+2})
\]
If all \( w_j > 0 \), then for \( f \in C^{r+1} \) with \( 0 \leq r \leq d \),

\[
|E(f)| \leq \frac{c_r}{d^{r+1}} \left( \frac{b-a}{2} \right)^{r+2} \| f^{(r+1)} \|_{\infty}
\]

\( c_r \) does not depend on \( Q \).
If all $w_j > 0$, then for $f \in C^{r+1}$ with $0 \leq r \leq d$,

$$|E(f)| \leq \frac{c_r}{d^{r+1}} \left( \frac{b-a}{2} \right)^{r+2} \|f^{(r+1)}\|_{\infty}$$

$c_r$ does not depend on $Q$.

According to this bound,

- Even for $f$ of modest differentiability, splitting $(\alpha_j, \beta_j)$ reduces $E_j(f)$.
- A formula with bigger $d$ is more accurate.
- The bigger $r$ is, the better.
Peano Kernel

If \( f \in C^{r+1}[a, b] \), one form of Taylor’s theorem is

\[
 f(x) = f(a) + \ldots + \frac{f^{(r)}(a)}{r!} (x - a)^r + \frac{1}{r!} \int_a^b f^{(r+1)}(t) (x - t)^r_+ \, dt
\]

Using \( E(T_r) = 0 \) for polynomials \( T_r(x) \) of degree \( r \leq d \),

\[
 E(f) = \int_a^b K_r(t) f^{(r+1)}(t) \, dt
\]

where

\[
 K_r(t) = \frac{(b - t)^{r+1}}{(r + 1)!} \frac{1}{r!} \sum_{j=1}^M w_j (x_j - t)^r_+
\]
Accuracy

After rescaling to $[-1, 1]$, we get the bound

$$|E(f)| \leq \left( \int_{-1}^{+1} K_r^2(t) \, dt \right)^{1/2} \left( \frac{b - a}{2} \right)^{r+2} \|f^{(r+1)}\|_2$$

$K_r^2(t)$ is a polynomial of degree $2r + 2$ on the subintervals $(-1, x_1), (x_1, x_2), \ldots, (x_{M-1}, x_M), (x_M, +1)$. A Gauss-Legendre formula with $r + 2$ nodes is exact.

For the compound trapezoidal rule of $M$ nodes,

$$\|K_1\|_2 = \frac{2}{\sqrt{15}} \frac{1}{(M - 1)^2}$$
\[ \| K_r \|_2 \text{ for } f \in C^{r+1} \]

<table>
<thead>
<tr>
<th>r</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>quad</td>
<td>4.7e-01</td>
<td>8.6e-02</td>
<td>2.3e-02</td>
<td>1.0e-02</td>
</tr>
<tr>
<td>quadl</td>
<td>3.2e-01</td>
<td>3.7e-02</td>
<td>5.6e-03</td>
<td>1.0e-03</td>
</tr>
<tr>
<td>quadgk</td>
<td>1.4e-01</td>
<td>6.8e-03</td>
<td>4.2e-04</td>
<td>2.7e-05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>quadl</td>
<td>2.3e-04</td>
<td>8.8e-05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quadgk</td>
<td>1.9e-06</td>
<td>1.3e-07</td>
<td>9.8e-09</td>
<td>7.7e-10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>quadgk</td>
<td>6.4e-11</td>
<td>5.9e-12</td>
<td>5.9e-13</td>
<td>7.0e-14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>quadgk</td>
<td>1.0e-14</td>
<td>2.7e-15</td>
</tr>
</tbody>
</table>
\[ \| K^*_r \|_2 / \| K_r \|_2 \text{ for } f \in C^{r+1} \]

<table>
<thead>
<tr>
<th>( r )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>quad</td>
<td>0.5375</td>
<td>0.2981</td>
<td>0.1667</td>
<td>0.0679</td>
<td>0.0140</td>
<td>0.0029</td>
<td></td>
<td></td>
<td>1.0e-3</td>
<td>3.9e-4</td>
<td>1.4e-4</td>
<td>4.3e-5</td>
</tr>
<tr>
<td>quadl</td>
<td>0.5000</td>
<td>0.2379</td>
<td>0.1061</td>
<td>0.0424</td>
<td>0.0276</td>
<td>0.0127</td>
<td>0.0057</td>
<td>0.0025</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quadgk</td>
<td>0.4987</td>
<td>0.2466</td>
<td>0.1206</td>
<td>0.0582</td>
<td>0.0276</td>
<td>0.0127</td>
<td>0.0057</td>
<td>0.0025</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.1e-5</td>
<td>1.6e-6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Peano Analysis of MATLAB Pairs

- Even when the integrand is of low differentiability,
  - increasing $d$ increases the accuracy and
  - all the pairs have acceptable error estimates.
Peano Analysis of MATLAB Pairs

- Even when the integrand is of low differentiability,
  - increasing $d$ increases the accuracy and
  - all the pairs have acceptable error estimates.

- With even modest differentiability, \texttt{quadgk}
  - is much more accurate than the other pairs and
  - has very good error estimates.
Infinite Intervals

quadl does not provide for infinite intervals.
\textbf{Infinite Intervals}

\texttt{quadl} does not provide for infinite intervals.

\texttt{quadgk} requires \( a < b \), but \( \pm \infty \) are allowed. Algebraic transformations are used to get a finite interval. The new integrand has end point singularities. For example,

\[
\int_0^\infty f(x) \, dx = \int_0^1 f(\phi(t)) \phi'(t) \, dt = \int_0^1 g(t) \, dt
\]

where \( \phi(t) = t/(1 - t) \) and \( \phi'(t) = 1/(1 - t)^2 \).
Infinite Intervals

quadl does not provide for infinite intervals.

quadgk requires $a < b$, but $\pm \infty$ are allowed. Algebraic transformations are used to get a finite interval. The new integrand has end point singularities. For example,

$$\int_{0}^{\infty} f(x) \, dx = \int_{0}^{1} f(\phi(t)) \phi'(t) \, dt = \int_{0}^{1} g(t) \, dt$$

where $\phi(t) = t / (1 - t)$ and $\phi'(t) = 1 / (1 - t)^2$.

For the integral to exist, we must have $f(x) \to 0$ pretty rapidly as $x \to \infty$. If, say, $f(x) \sim c e^{-x} x^k$, then $g(t) \to 0$ as $t \to 1$ and the singularity is only apparent.
Split $[a, b]$ into subintervals so that any singularity is at an end point and treat the subintervals separately.
Singularities

Split \([a, b]\) into subintervals so that any singularity is at an end point and treat the subintervals separately.

Function `quad` and `quadl` evaluate \(f(a)\) and \(f(b)\), so the original versions generally failed. Nowadays they trap `Inf` and `NaN` and avoid trouble by shrinking \([a, b]\) slightly. This works surprisingly well.
Singularities

Split \([a, b]\) into subintervals so that any singularity is at an end point and treat the subintervals separately.

\texttt{quad} and \texttt{quadl} evaluate \(f(a)\) and \(f(b)\), so the original versions generally failed. Nowadays they trap \texttt{Inf} and \texttt{NaN} and avoid trouble by shrinking \([a, b]\) slightly. This works surprisingly well.

\texttt{quadgk} does not evaluate \(f(x)\) at the end points, so there is no formal difficulty. It changes variables so as to \texttt{weaken} end point singularities. This reduces the effective degree of precision for the subintervals that include \(a\) and \(b\), but \(d\) is big enough that this is unimportant.
Change of Variable

To illustrate the changes of variable,

\[ \int_0^x f(x) \, dx = \int_0^t f(t^2) \cdot 2t \, dt = \int_0^t g(t) \, dt \]
Change of Variable

To illustrate the changes of variable,

\[ \int_0^1 f(x) \, dx = \int_0^1 f(t^2) 2t \, dt = \int_0^1 g(t) \, dt \]

If \( f(x) \sim c \log x \) as \( x \to 0 \), then \( g(t) \sim 4c t \log t \) as \( t \to 0 \). The new integrand is not singular.
To illustrate the changes of variable,

\[ \int_0^1 f(x) \, dx = \int_0^1 f(t^2) \, 2t \, dt = \int_0^1 g(t) \, dt \]

If \( f(x) \sim c \log x \) as \( x \to 0 \), then \( g(t) \sim 4ct \log t \) as \( t \to 0 \).

The new integrand is not singular.

If \( f(x) \sim cx^\alpha \) with \( \alpha > -1 \), then \( g(t) \sim 2ct^{2\alpha+1} \).

The new integrand has a weaker singularity at the endpoint. Indeed, if \( \alpha > -1/2 \), it is not singular.
Example

Approximate

\[ \int_0^\infty e^{-x^2} (\log(x))^2 \, dx \]

Note vectorized evaluation of the integrand:

\[
\begin{align*}
\text{f} &= @(x) \exp(-x.^2).*\log(x).^2; \\
\text{Q} &= \text{quadgk}([\text{f},0,\text{Inf}]) \\
&\quad \% \text{Using defaults here--equivalent to} \\
&\quad \% Q = \text{quadgk}([\text{f},0,\text{Inf},...) \\
&\quad \% \quad 'AbsTol',1e-10,'RelTol',1e-6)
\end{align*}
\]

\text{quadl} \text{ does not control relative error.}

The relative error of \( Q \approx 1.9475 \) is \( 2 \times 10^{-8} \).
Closing Remarks

`quadgk` has other capabilities not found in `quadl`:

- Piecewise-smooth integrands can be integrated effectively using a vector of `Waypoints`.

- Complex-valued functions can be integrated over polygonal paths in the complex plane specified by `Waypoints`. 
Closing Remarks

**quadgk** has other capabilities not found in **quadl**

- Piecewise-smooth integrands can be integrated effectively using a vector of **Waypoints**.
- Complex-valued functions can be integrated over polygonal paths in the complex plane specified by **Waypoints**.

Extensive numerical testing shows that in addition to being more capable, **quadgk** is much faster and much more reliable than **quadl**.