Scientific Computing: Numerical Integration

Aleksandar Donev
Courant Institute, NYU[1]
donev@courant.nyu.edu

[1]Course MATH-GA.2043 or CSCI-GA.2112, Fall 2020

Nov 5th, 2020

Outline

Outline

Numerical Quadrature

- We want to numerically approximate a definite integral

$$J = \int_a^b f(x)dx.$$

Numerical Quadrature

- We want to numerically approximate a definite integral

$$J = \int_a^b f(x)dx.$$

- The function $f(x)$ may not have a closed-form integral, or it may itself not be in closed form.

Numerical Quadrature

- We want to numerically approximate a definite integral

$$J = \int_a^b f(x)dx.$$

- The function $f(x)$ may not have a closed-form integral, or it may itself not be in closed form.
- Recall that the integral gives the area under the curve $f(x)$, and also the **Riemann sum**:

$$\lim_{n\to\infty} \sum_{i=0}^{n} f(t_i)(x_{i+1} - x_i) = J, \text{ where } x_i \le t_i \le x_{i+1}$$

Numerical Quadrature

- We want to numerically approximate a definite integral

$$J = \int_a^b f(x)dx.$$

- The function $f(x)$ may not have a closed-form integral, or it may itself not be in closed form.
- Recall that the integral gives the area under the curve $f(x)$, and also the **Riemann sum**:

$$\lim_{n\to\infty} \sum_{i=0}^{n} f(t_i)(x_{i+1} - x_i) = J, \text{ where } x_i \leq t_i \leq x_{i+1}$$

- A **quadrature formula** approximates the Riemann integral as a **discrete sum** over a set of n nodes:

$$J \approx J_n = \sum_{i=1}^{n} \alpha_i f(x_i)$$

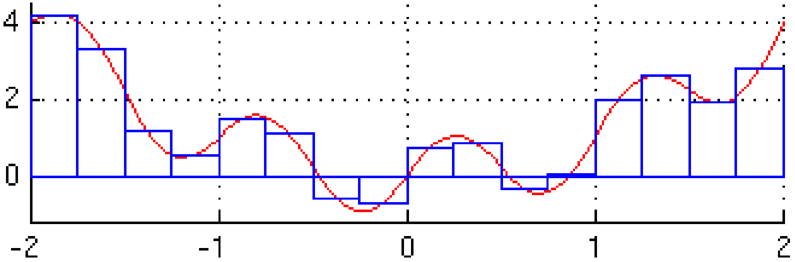
Midpoint Quadrature

Split the interval into n intervals of width $h = (b - a)/n$ (**stepsize**), and then take as the nodes the midpoint of each interval:

Midpoint Quadrature

Split the interval into n intervals of width $h = (b-a)/n$ (**stepsize**), and then take as the nodes the midpoint of each interval:

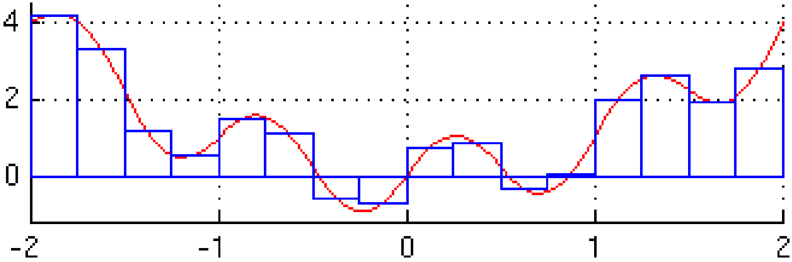
$$x_k = a + (2k-1)h/2, \quad k = 1, \ldots, n$$



Midpoint Quadrature

Split the interval into n intervals of width $h = (b - a)/n$ (**stepsize**), and then take as the nodes the midpoint of each interval:

$$x_k = a + (2k - 1)h/2, \quad k = 1, \dots, n$$



$$J_n = h \sum_{k=1}^{n} f(x_k), \text{ and clearly } \lim_{n \to \infty} J_n = J$$

Quadrature Error

- Focus on one of the sub intervals, and estimate the quadrature error using the midpoint rule assuming $f(x) \in C^{(2)}$:

Quadrature Error

- Focus on one of the sub intervals, and estimate the quadrature error using the midpoint rule assuming $f(x) \in C^{(2)}$:

$$\varepsilon^{(i)} = \left[\int_{x_i - h/2}^{x_i + h/2} f(x)dx \right] - hf(x_i)$$

Quadrature Error

- Focus on one of the sub intervals, and estimate the quadrature error using the midpoint rule assuming $f(x) \in C^{(2)}$:

$$\varepsilon^{(i)} = \left[\int_{x_i - h/2}^{x_i + h/2} f(x) dx \right] - hf(x_i)$$

- Expanding $f(x)$ into a Taylor series around x_i to first order,

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2} f'' \left[\eta(x) \right] (x - x_i)^2,$$

Quadrature Error

- Focus on one of the sub intervals, and estimate the quadrature error using the midpoint rule assuming $f(x) \in C^{(2)}$:

$$\varepsilon^{(i)} = \left[\int_{x_i-h/2}^{x_i+h/2} f(x)dx\right] - hf(x_i)$$

- Expanding $f(x)$ into a Taylor series around x_i to first order,

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''\left[\eta(x)\right](x - x_i)^2,$$

The linear term integrates to zero, so we get

$$\int_{x_i-h/2}^{x_i+h/2} f'(x_i)(x - x_i) = 0 \quad \Rightarrow$$

$$\varepsilon^{(i)} = \frac{1}{2}\int_{x_i-h/2}^{x_i+h/2} f''\left[\eta(x)\right](x - x_i)^2 dx$$

Composite Quadrature Error

- Using a generalized mean value theorem we can show

$$\varepsilon^{(i)} = f''[\xi] \frac{1}{2} \int_h (x - x_i)^2 dx = \frac{h^3}{24} f''[\xi] \quad \text{for some } a < \xi < b$$

- Now, combining the errors from all of the intervals together gives the **global error**

Composite Quadrature Error

- Using a generalized mean value theorem we can show

$$\varepsilon^{(i)} = f''\left[\xi\right]\frac{1}{2}\int_h (x-x_i)^2 dx = \frac{h^3}{24}f''\left[\xi\right] \quad \text{for some } a<\xi<b$$

- Now, combining the errors from all of the intervals together gives the **global error**

$$\varepsilon = \int_a^b f(x)dx - h\sum_{k=1}^{n} f(x_k) = J - J_n = \frac{h^3}{24}\sum_{k=1}^{n} f''\left[\xi_k\right]$$

Composite Quadrature Error

- Using a generalized mean value theorem we can show

$$\varepsilon^{(i)} = f''\left[\xi\right]\frac{1}{2}\int_h (x - x_i)^2 dx = \frac{h^3}{24} f''\left[\xi\right] \quad \text{for some } a < \xi < b$$

- Now, combining the errors from all of the intervals together gives the **global error**

$$\varepsilon = \int_a^b f(x)dx - h\sum_{k=1}^{n} f(x_k) = J - J_n = \frac{h^3}{24}\sum_{k=1}^{n} f''\left[\xi_k\right]$$

- Use a discrete generalization of the mean value theorem to prove **second-order accuracy**

$$\varepsilon = \frac{h^3}{24} n \left(f''\left[\xi\right]\right) = \frac{b-a}{24} \cdot h^2 \cdot f''\left[\xi\right] \qquad \text{for some } a < \xi < b$$

Interpolatory Quadrature

Instead of integrating $f(x)$, integrate a polynomial interpolant $\phi(x) \approx f(x)$:

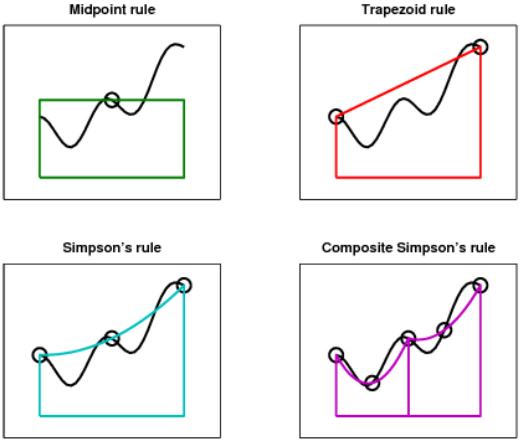


Figure 6.2. *Four quadrature rules.*

Trapezoidal Rule

- Consider integrating an **interpolating function** $\phi(x)$ which passes through $n+1$ **nodes** x_i:

$$\phi(x_i) = y_i = f(x_i) \text{ for } i = 0, 2, \ldots, m.$$

Trapezoidal Rule

- Consider integrating an **interpolating function** $\phi(x)$ which passes through $n+1$ **nodes** x_i:

$$\phi(x_i) = y_i = f(x_i) \text{ for } i = 0, 2, \dots, m.$$

- First take the **piecewise linear interpolant** and integrate it over the sub-interval $I_i = [x_{i-1}, x_i]$:

$$\phi_i^{(1)}(x) = y_{i-1} + \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x - x_i)$$

Trapezoidal Rule

- Consider integrating an **interpolating function** $\phi(x)$ which passes through $n+1$ **nodes** x_i:

$$\phi(x_i) = y_i = f(x_i) \text{ for } i = 0, 2, \ldots, m.$$

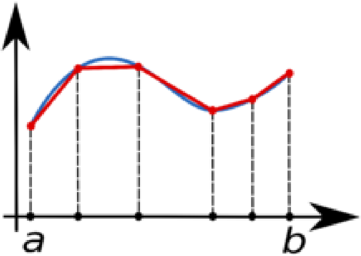
- First take the **piecewise linear interpolant** and integrate it over the sub-interval $I_i = [x_{i-1}, x_i]$:

$$\phi_i^{(1)}(x) = y_{i-1} + \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x - x_i)$$

to get the **trapezoidal formula** (the area of a trapezoid):

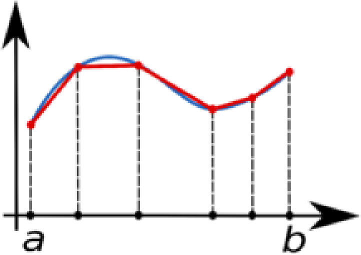
$$\int_{x \in I_i} \phi_i^{(1)}(x)dx = h\frac{f(x_{i-1}) + f(x_i)}{2}$$

Composite Trapezoidal Rule



- Now add the integrals over all of the sub-intervals we get the **composite trapezoidal quadrature rule**:

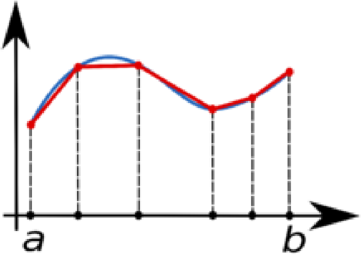
Composite Trapezoidal Rule



- Now add the integrals over all of the sub-intervals we get the **composite trapezoidal quadrature rule**:

$$\int_a^b f(x)dx \approx \frac{h}{2}\sum_{i=1}^{n}\left[f(x_{i-1}) + f(x_i)\right]$$
$$= \frac{h}{2}\left[f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)\right]$$

Composite Trapezoidal Rule



- Now add the integrals over all of the sub-intervals we get the **composite trapezoidal quadrature rule**:

$$\int_a^b f(x)dx \approx \frac{h}{2}\sum_{i=1}^{n}\left[f(x_{i-1}) + f(x_i)\right]$$
$$= \frac{h}{2}\left[f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)\right]$$

with similar error to the midpoint rule.

Simpson's Quadrature Formula

- As for the midpoint rule, split the interval into n intervals of width $h = (b - a)/n$, and then take as the nodes the endpoints and midpoint of each interval:

Simpson's Quadrature Formula

- As for the midpoint rule, split the interval into n intervals of width $h = (b - a)/n$, and then take as the nodes the endpoints and midpoint of each interval:

$$x_k = a + kh, \quad k = 0, \ldots, n$$
$$\bar{x}_k = a + (2k - 1)h/2, \quad k = 1, \ldots, n$$

Simpson's Quadrature Formula

- As for the midpoint rule, split the interval into n intervals of width $h = (b-a)/n$, and then take as the nodes the endpoints and midpoint of each interval:

$$x_k = a + kh, \quad k = 0, \ldots, n$$
$$\bar{x}_k = a + (2k-1)h/2, \quad k = 1, \ldots, n$$

- Then, take the **piecewise quadratic interpolant** $\phi_i(x)$ in the sub-interval $I_i = [x_{i-1}, x_i]$ to be the parabola passing through the nodes (x_{i-1}, y_{i-1}), (x_i, y_i), and $(\bar{x}_i, \bar{y}_i)$.

Simpson's Quadrature Formula

- As for the midpoint rule, split the interval into n intervals of width $h = (b-a)/n$, and then take as the nodes the endpoints and midpoint of each interval:

$$
\begin{aligned}
x_k &= a + kh, \quad k = 0, \ldots, n \\
\bar{x}_k &= a + (2k-1)h/2, \quad k = 1, \ldots, n
\end{aligned}
$$

- Then, take the **piecewise quadratic interpolant** $\phi_i(x)$ in the sub-interval $I_i = [x_{i-1}, x_i]$ to be the parabola passing through the nodes (x_{i-1}, y_{i-1}), (x_i, y_i), and $(\bar{x}_i, \bar{y}_i)$.
- Integrating this interpolant in each interval and summing gives the **Simpson quadrature rule**:

$$
J_S = \frac{h}{6}\left[f(x_0) + 4f(\bar{x}_1) + 2f(x_1) + \cdots + 2f(x_{n-1}) + 4f(\bar{x}_n) + f(x_n)\right]
$$

Simpson's Quadrature Formula

- As for the midpoint rule, split the interval into n intervals of width $h = (b-a)/n$, and then take as the nodes the endpoints and midpoint of each interval:

$$x_k = a + kh, \quad k = 0, \dots, n$$
$$\bar{x}_k = a + (2k-1)h/2, \quad k = 1, \dots, n$$

- Then, take the **piecewise quadratic interpolant** $\phi_i(x)$ in the sub-interval $I_i = [x_{i-1}, x_i]$ to be the parabola passing through the nodes (x_{i-1}, y_{i-1}), (x_i, y_i), and $(\bar{x}_i, \bar{y}_i)$.
- Integrating this interpolant in each interval and summing gives the **Simpson quadrature rule**:

$$J_S = \frac{h}{6}\left[f(x_0) + 4f(\bar{x}_1) + 2f(x_1) + \cdots + 2f(x_{n-1}) + 4f(\bar{x}_n) + f(x_n)\right]$$

$$\varepsilon = J - J_s = -\frac{(b-a)}{2880} \cdot h^4 \cdot f^{(4)}(\xi).$$

Gauss Quadrature

- To reach **spectral accuracy** for **smooth functions**, instead of using higher-degree polynomial interpolants (recall Runge's phenomenon), let's try using n **non-equispaced nodes**:

$$J \approx J_n = \sum_{i=0}^{n} w_i f(x_i)$$

Gauss Quadrature

- To reach **spectral accuracy** for **smooth functions**, instead of using higher-degree polynomial interpolants (recall Runge's phenomenon), let's try using n **non-equispaced nodes**:

$$J \approx J_n = \sum_{i=0}^{n} w_i f(x_i)$$

- It can be shown that there is a special set of $n+1$ nodes and weights, so that the quadrature formula is exact for polynomials of degree up to $m = 2n - 1$,

$$\int_a^b p_m(x)dx = \sum_{i=0}^{n} w_i p_m(x_i).$$

Gauss Quadrature

- To reach **spectral accuracy** for **smooth functions**, instead of using higher-degree polynomial interpolants (recall Runge's phenomenon), let's try using n **non-equispaced nodes**:

$$J \approx J_n = \sum_{i=0}^{n} w_i f(x_i)$$

- It can be shown that there is a special set of $n+1$ nodes and weights, so that the quadrature formula is exact for polynomials of degree up to $m = 2n - 1$,

$$\int_a^b p_m(x)dx = \sum_{i=0}^{n} w_i p_m(x_i).$$

- This gives the **Gauss quadrature** based on the **Gauss nodes and weights**, usually pre-tabulated for the standard interval $[-1, 1]$:

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \sum_{i=0}^{n} w_i f(x_i).$$

Gauss Weights and Nodes

- The low-order Gauss formulas are:

$$n = 1: \int_{-1}^{1} f(x)dx \approx f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

$$n = 2: \int_{-1}^{1} f(x)dx \approx \frac{5}{9} f\left(-\frac{\sqrt{15}}{5}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\frac{\sqrt{15}}{5}\right)$$

Gauss Weights and Nodes

- The low-order Gauss formulas are:

$$n = 1 : \int_{-1}^{1} f(x)dx \approx f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

$$n = 2 : \int_{-1}^{1} f(x)dx \approx \frac{5}{9} f\left(-\frac{\sqrt{15}}{5}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\frac{\sqrt{15}}{5}\right)$$

- The weights and nodes are either **tabulated** or calculated to numerical precision **on the fly**, for example, using eigenvalue methods.

Gauss Weights and Nodes

- The low-order Gauss formulas are:

$$n=1: \int_{-1}^{1} f(x)dx \approx f\left(-\frac{1}{\sqrt{3}}\right)+f\left(\frac{1}{\sqrt{3}}\right)$$

$$n=2: \int_{-1}^{1} f(x)dx \approx \frac{5}{9}f\left(-\frac{\sqrt{15}}{5}\right)+\frac{8}{9}f(0)+\frac{5}{9}f\left(\frac{\sqrt{15}}{5}\right)$$

- The weights and nodes are either **tabulated** or calculated to numerical precision **on the fly**, for example, using eigenvalue methods.
- Gauss quadrature is **very accurate for smooth functions** even with few nodes.

Gauss Weights and Nodes

- The low-order Gauss formulas are:

$$n = 1 : \int_{-1}^{1} f(x)dx \approx f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

$$n = 2 : \int_{-1}^{1} f(x)dx \approx \frac{5}{9} f\left(-\frac{\sqrt{15}}{5}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\frac{\sqrt{15}}{5}\right)$$

- The weights and nodes are either **tabulated** or calculated to numerical precision **on the fly**, for example, using eigenvalue methods.
- Gauss quadrature is **very accurate for smooth functions** even with few nodes.
- The MATLAB function $quadl(f, a, b)$ uses (adaptive) Gauss-Lobatto quadrature.

Gauss Weights and Nodes

- The low-order Gauss formulas are:

$$n = 1 : \int_{-1}^{1} f(x)dx \approx f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

$$n = 2 : \int_{-1}^{1} f(x)dx \approx \frac{5}{9} f\left(-\frac{\sqrt{15}}{5}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\frac{\sqrt{15}}{5}\right)$$

- The weights and nodes are either **tabulated** or calculated to numerical precision **on the fly**, for example, using eigenvalue methods.
- Gauss quadrature is **very accurate for smooth functions** even with few nodes.
- The MATLAB function $quadl(f, a, b)$ uses (adaptive) Gauss-Lobatto quadrature.
- An alternative is to use Chebyshev nodes and weights, called **Clenshaw-Curtis quadrature** (exact for polynomials of degree n).

Outline

1. Numerical Integration in 1D
2. Adaptive / Refinement Methods
3. Higher Dimensions
4. Conclusions

Asymptotic Error Expansions

- The idea in **Richardson extrapolation** is to use an error estimate formula to **extrapolate a more accurate answer** from less-accurate answers.

Asymptotic Error Expansions

- The idea in **Richardson extrapolation** is to use an error estimate formula to **extrapolate a more accurate answer** from less-accurate answers.
- Assume that we have a quadrature formula for which we have a theoretical error estimate:

$$J_h = \sum_{i=1}^{n} \alpha_i f(x_i) = J + \alpha h^p + O\left(h^{p+1}\right)$$

Asymptotic Error Expansions

- The idea in **Richardson extrapolation** is to use an error estimate formula to **extrapolate a more accurate answer** from less-accurate answers.
- Assume that we have a quadrature formula for which we have a theoretical error estimate:

$$J_h = \sum_{i=1}^{n} \alpha_i f(x_i) = J + \alpha h^p + O\left(h^{p+1}\right)$$

- Recall the **big O notation**: $g(x) = O\left(h^p\right)$ if:

$$\exists \left(h_0, C\right) > 0 \text{ s.t. } \left|g(x)\right| < C \left|h\right|^p \text{ whenever } \left|h\right| < h_0$$

Asymptotic Error Expansions

- The idea in **Richardson extrapolation** is to use an error estimate formula to **extrapolate a more accurate answer** from less-accurate answers.
- Assume that we have a quadrature formula for which we have a theoretical error estimate:

$$J_h = \sum_{i=1}^{n} \alpha_i f(x_i) = J + \alpha h^p + O\left(h^{p+1}\right)$$

- Recall the **big O notation**: $g(x) = O(h^p)$ if:

$$\exists \left(h_0, C\right) > 0 \text{ s.t. } \left|g(x)\right| < C \left|h\right|^p \text{ whenever } \left|h\right| < h_0$$

- For trapezoidal formula

$$\varepsilon = \frac{b-a}{24} \cdot h^2 \cdot f''[\xi] = O\left(h^2\right).$$

Richardson Extrapolation

- Now repeat the calculation but with step size $2h$ (for equi-spaced nodes just skip the odd nodes):

Richardson Extrapolation

- Now repeat the calculation but with step size $2h$ (for equi-spaced nodes just skip the odd nodes):

$$\tilde{J}(h) = J + \alpha h^p + O\left(h^{p+1}\right)$$
$$\tilde{J}(2h) = J + \alpha 2^p h^p + O\left(h^{p+1}\right)$$

Richardson Extrapolation

- Now repeat the calculation but with step size $2h$ (for equi-spaced nodes just skip the odd nodes):

$$\tilde{J}(h) = J + \alpha h^p + O\left(h^{p+1}\right)$$
$$\tilde{J}(2h) = J + \alpha 2^p h^p + O\left(h^{p+1}\right)$$

- Solve for α and obtain

$$J = \frac{2^p \tilde{J}(h) - \tilde{J}(2h)}{2^p - 1} + O\left(h^{p+1}\right),$$

Richardson Extrapolation

- Now repeat the calculation but with step size $2h$ (for equi-spaced nodes just skip the odd nodes):

$$\tilde{J}(h) = J + \alpha h^p + O\left(h^{p+1}\right)$$
$$\tilde{J}(2h) = J + \alpha 2^p h^p + O\left(h^{p+1}\right)$$

- Solve for α and obtain

$$J = \frac{2^p \tilde{J}(h) - \tilde{J}(2h)}{2^p - 1} + O\left(h^{p+1}\right),$$

which now has order of accuracy $p+1$ instead of p.

Richardson Extrapolation

- Now repeat the calculation but with step size $2h$ (for equi-spaced nodes just skip the odd nodes):

$$\tilde{J}(h) = J + \alpha h^p + O\left(h^{p+1}\right)$$
$$\tilde{J}(2h) = J + \alpha 2^p h^p + O\left(h^{p+1}\right)$$

- Solve for α and obtain

$$J = \frac{2^p \tilde{J}(h) - \tilde{J}(2h)}{2^p - 1} + O\left(h^{p+1}\right),$$

which now has order of accuracy $p+1$ instead of p.

- The composite trapezoidal quadrature gives $\tilde{J}(h)$ with order of accuracy $p = 2$, $\tilde{J}(h) = J + O\left(h^2\right)$.

Romberg Quadrature

- Assume that we have evaluated $f(x)$ at $n = 2^m + 1$ equi-spaced nodes, $h = 2^{-m}(b-a)$, giving approximation $\tilde{J}(h)$.

Romberg Quadrature

- Assume that we have evaluated $f(x)$ at $n = 2^m + 1$ equi-spaced nodes, $h = 2^{-m}(b - a)$, giving approximation $\tilde{J}(h)$.
- We can also easily compute $\tilde{J}(2h)$ by simply skipping the odd nodes. And also $\tilde{J}(4h)$, and in general, $\tilde{J}(2^q h)$, $q = 0, \ldots, m$.

Romberg Quadrature

- Assume that we have evaluated $f(x)$ at $n = 2^m + 1$ equi-spaced nodes, $h = 2^{-m}(b - a)$, giving approximation $\tilde{J}(h)$.
- We can also easily compute $\tilde{J}(2h)$ by simply skipping the odd nodes. And also $\tilde{J}(4h)$, and in general, $\tilde{J}(2^q h)$, $q = 0, \ldots, m$.
- We can keep applying **Richardson extrapolation recursively** to get **Romberg's quadrature**:

Romberg Quadrature

- Assume that we have evaluated $f(x)$ at $n = 2^m + 1$ equi-spaced nodes, $h = 2^{-m}(b-a)$, giving approximation $\tilde{J}(h)$.
- We can also easily compute $\tilde{J}(2h)$ by simply skipping the odd nodes. And also $\tilde{J}(4h)$, and in general, $\tilde{J}(2^q h)$, $q = 0, \ldots, m$.
- We can keep applying **Richardson extrapolation recursively** to get **Romberg's quadrature**:
  Combine $\tilde{J}(2^q h)$ and $\tilde{J}(2^{q-1} h)$ to get a better estimate. Then combine the estimates to get an even better estimates, etc.

Romberg Quadrature

- Assume that we have evaluated $f(x)$ at $n = 2^m + 1$ equi-spaced nodes, $h = 2^{-m}(b-a)$, giving approximation $\tilde{J}(h)$.
- We can also easily compute $\tilde{J}(2h)$ by simply skipping the odd nodes. And also $\tilde{J}(4h)$, and in general, $\tilde{J}(2^q h)$, $q = 0, \ldots, m$.
- We can keep applying **Richardson extrapolation recursively** to get **Romberg's quadrature**:
  Combine $\tilde{J}(2^q h)$ and $\tilde{J}(2^{q-1} h)$ to get a better estimate. Then combine the estimates to get an even better estimates, etc.

$$J_{r,0} = \tilde{J}\left(\frac{b-a}{2^r}\right), \quad r = 0, \ldots, m$$

Romberg Quadrature

- Assume that we have evaluated $f(x)$ at $n = 2^m + 1$ equi-spaced nodes, $h = 2^{-m}(b - a)$, giving approximation $\tilde{J}(h)$.
- We can also easily compute $\tilde{J}(2h)$ by simply skipping the odd nodes. And also $\tilde{J}(4h)$, and in general, $\tilde{J}(2^q h)$, $q = 0, \ldots, m$.
- We can keep applying **Richardson extrapolation recursively** to get **Romberg's quadrature**:
  Combine $\tilde{J}(2^q h)$ and $\tilde{J}(2^{q-1} h)$ to get a better estimate. Then combine the estimates to get an even better estimates, etc.

$$J_{r,0} = \tilde{J}\left(\frac{b-a}{2^r}\right), \quad r = 0, \ldots, m$$

$$J_{r,q+1} = \frac{4^{q+1} J_{r,q} - J_{r-1,q}}{4^{q+1} - 1}, \quad q = 0, \ldots, m-1, \quad r = q+1, \ldots, m$$

Romberg Quadrature

- Assume that we have evaluated $f(x)$ at $n = 2^m + 1$ equi-spaced nodes, $h = 2^{-m}(b - a)$, giving approximation $\tilde{J}(h)$.
- We can also easily compute $\tilde{J}(2h)$ by simply skipping the odd nodes. And also $\tilde{J}(4h)$, and in general, $\tilde{J}(2^q h)$, $q = 0, \ldots, m$.
- We can keep applying **Richardson extrapolation recursively** to get **Romberg's quadrature**:
  Combine $\tilde{J}(2^q h)$ and $\tilde{J}(2^{q-1} h)$ to get a better estimate. Then combine the estimates to get an even better estimates, etc.

$$J_{r,0} = \tilde{J}\left(\frac{b-a}{2^r}\right), \quad r = 0, \ldots, m$$

$$J_{r,q+1} = \frac{4^{q+1} J_{r,q} - J_{r-1,q}}{4^{q+1} - 1}, \quad q = 0, \ldots, m-1, \quad r = q+1, \ldots, m$$

- The final answer, $J_{m,m} = J + O\left(h^{2(m+1)}\right)$ is much more accurate than the starting $J_{m,0} = J + O\left(h^2\right)$, for **smooth** functions.

Adaptive (Automatic) Integration

- We would like a way to control the error of the integration, that is, specify a **target error** ε_{max} and let the algorithm figure out the correct step size h to satisfy

Adaptive (Automatic) Integration

- We would like a way to control the error of the integration, that is, specify a **target error** ε_{max} and let the algorithm figure out the correct step size h to satisfy

$$|\varepsilon| \lesssim \varepsilon_{max},$$

where ε is an **error estimate**.

Adaptive (Automatic) Integration

- We would like a way to control the error of the integration, that is, specify a **target error** ε_{max} and let the algorithm figure out the correct step size h to satisfy

$$|\varepsilon| \lesssim \varepsilon_{max},$$

  where ε is an **error estimate**.

- Importantly, h may **vary adaptively** in different parts of the integration interval:

Adaptive (Automatic) Integration

- We would like a way to control the error of the integration, that is, specify a **target error** ε_{max} and let the algorithm figure out the correct step size h to satisfy

$$|\varepsilon| \lesssim \varepsilon_{max},$$

  where ε is an **error estimate**.
- Importantly, h may **vary adaptively** in different parts of the integration interval:
  Smaller step size when the function has larger derivatives.

Adaptive (Automatic) Integration

- We would like a way to control the error of the integration, that is, specify a **target error** ε_{max} and let the algorithm figure out the correct step size h to satisfy

$$|\varepsilon| \lesssim \varepsilon_{max},$$

  where ε is an **error estimate**.
- Importantly, h may **vary adaptively** in different parts of the integration interval:
  Smaller step size when the function has larger derivatives.
- The crucial step is obtaining an error estimate: Use the same idea as in Richardson extrapolation.

Error Estimate for Simpson's Quadrature

- Assume we are using Simpson's quadrature and compute the integral $J(h)$ with step size h.

Error Estimate for Simpson's Quadrature

- Assume we are using Simpson's quadrature and compute the integral $J(h)$ with step size h.
- Then also compute integrals for the left half and for the right half with step size $h/2$, $J(h/2) = J_L(h/2) + J_R(h/2)$.

Error Estimate for Simpson's Quadrature

- Assume we are using Simpson's quadrature and compute the integral $J(h)$ with step size h.
- Then also compute integrals for the left half and for the right half with step size $h/2$, $J(h/2) = J_L(h/2) + J_R(h/2)$.

$$J = J(h) - \frac{1}{2880} \cdot h^5 \cdot f^{(4)}(\xi)$$

$$J = J(h/2) - \frac{1}{2880} \cdot \frac{h^4}{32} \cdot \left[f^{(4)}(\xi_L) + f^{(4)}(\xi_R) \right].$$

Error Estimate for Simpson's Quadrature

- Assume we are using Simpson's quadrature and compute the integral $J(h)$ with step size h.
- Then also compute integrals for the left half and for the right half with step size $h/2$, $J(h/2) = J_L(h/2) + J_R(h/2)$.

$$J = J(h) - \frac{1}{2880} \cdot h^5 \cdot f^{(4)}(\xi)$$

$$J = J(h/2) - \frac{1}{2880} \cdot \frac{h^4}{32} \cdot \left[f^{(4)}(\xi_L) + f^{(4)}(\xi_R) \right].$$

- Now assume that the fourth derivative varies little over the interval, $f^{(4)}(\xi_L) \approx f^{(4)}(\xi_L) \approx f^{(4)}(\xi)$, to estimate:

Error Estimate for Simpson's Quadrature

- Assume we are using Simpson's quadrature and compute the integral $J(h)$ with step size h.
- Then also compute integrals for the left half and for the right half with step size $h/2$, $J(h/2) = J_L(h/2) + J_R(h/2)$.

$$J = J(h) - \frac{1}{2880} \cdot h^5 \cdot f^{(4)}(\xi)$$

$$J = J(h/2) - \frac{1}{2880} \cdot \frac{h^4}{32} \cdot \left[f^{(4)}(\xi_L) + f^{(4)}(\xi_R) \right].$$

- Now assume that the fourth derivative varies little over the interval, $f^{(4)}(\xi_L) \approx f^{(4)}(\xi_L) \approx f^{(4)}(\xi)$, to estimate:

$$\frac{1}{2880} \cdot h^5 \cdot f^{(4)}(\xi) \approx \frac{16}{15} [J(h) - J(h/2)]$$

$$J(h/2) - J \approx \varepsilon = \frac{1}{16} [J(h) - J(h/2)].$$

Adaptive Integration

- Now assume that we have split the integration interval $[a, b]$ into sub-intervals, and we are considering computing the integral over the sub-interval $[\alpha, \beta]$, with stepsize

$$h = \beta - \alpha.$$

Adaptive Integration

- Now assume that we have split the integration interval $[a, b]$ into sub-intervals, and we are considering computing the integral over the sub-interval $[\alpha, \beta]$, with stepsize

$$h = \beta - \alpha.$$

- We need to compute this sub-integral with accuracy

$$|\varepsilon(\alpha, \beta)| = \frac{1}{16} |[J(h) - J(h/2)]| \leq \varepsilon \frac{h}{b-a}.$$

Adaptive Integration

- Now assume that we have split the integration interval $[a, b]$ into sub-intervals, and we are considering computing the integral over the sub-interval $[\alpha, \beta]$, with stepsize

$$h = \beta - \alpha.$$

- We need to compute this sub-integral with accuracy

$$|\varepsilon(\alpha, \beta)| = \frac{1}{16} |[J(h) - J(h/2)]| \leq \varepsilon \frac{h}{b-a}.$$

- An adaptive integration algorithm is $J \approx J(a, b, \epsilon)$ where the **recursive function** is:

Adaptive Integration

- Now assume that we have split the integration interval $[a, b]$ into sub-intervals, and we are considering computing the integral over the sub-interval $[\alpha, \beta]$, with stepsize

$$h = \beta - \alpha.$$

- We need to compute this sub-integral with accuracy

$$|\varepsilon(\alpha, \beta)| = \frac{1}{16} \left|[J(h) - J(h/2)]\right| \leq \varepsilon \frac{h}{b-a}.$$

- An adaptive integration algorithm is $J \approx J(a, b, \epsilon)$ where the **recursive function** is:

$$J(\alpha, \beta, \epsilon) = \begin{cases} J(h/2) & \text{if } |J(h) - J(h/2)| \leq 16\varepsilon \\ J(\alpha, \frac{\alpha+\beta}{2}, \frac{\epsilon}{2}) + J(\frac{\alpha+\beta}{2}, \beta, \frac{\epsilon}{2}) & \text{otherwise} \end{cases}$$

Adaptive Integration

- Now assume that we have split the integration interval $[a, b]$ into sub-intervals, and we are considering computing the integral over the sub-interval $[\alpha, \beta]$, with stepsize

$$h = \beta - \alpha.$$

- We need to compute this sub-integral with accuracy

$$|\varepsilon(\alpha, \beta)| = \frac{1}{16} \left|[J(h) - J(h/2)]\right| \leq \varepsilon \frac{h}{b-a}.$$

- An adaptive integration algorithm is $J \approx J(a, b, \epsilon)$ where the **recursive function** is:

$$J(\alpha, \beta, \epsilon) = \begin{cases} J(h/2) & \text{if } |J(h) - J(h/2)| \leq 16\varepsilon \\ J(\alpha, \frac{\alpha+\beta}{2}, \frac{\epsilon}{2}) + J(\frac{\alpha+\beta}{2}, \beta, \frac{\epsilon}{2}) & \text{otherwise} \end{cases}$$

- In practice one also stops the refinement if $h < h_{min}$ and is more conservative e.g., use 10 instead of 16.

Piecewise constant / linear basis functions

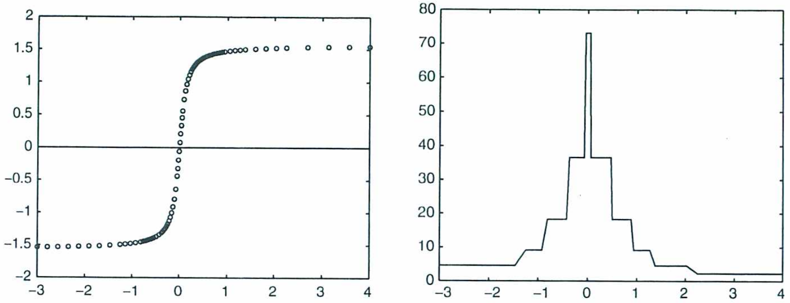


Fig. 9.4. Distribution of quadrature nodes (*left*); density of the integration stepsize in the approximation of the integral of Example 9.9 (*right*)

Outline

Regular Grids in Two Dimensions

- A **separable integral** can be done by doing integration along one axes first, then another:

Regular Grids in Two Dimensions

- A **separable integral** can be done by doing integration along one axes first, then another:

$$J = \int_{x=0}^{1} \int_{y=0}^{1} f(x,y)dxdy = \int_{x=0}^{1} dx \left[\int_{y=0}^{1} f(x,y)dy \right]$$

Regular Grids in Two Dimensions

- A **separable integral** can be done by doing integration along one axes first, then another:

$$J = \int_{x=0}^{1} \int_{y=0}^{1} f(x,y)dxdy = \int_{x=0}^{1} dx \left[\int_{y=0}^{1} f(x,y)dy \right]$$

- Consider evaluating the function at nodes on a **regular grid**

$$\mathbf{x}_{i,j} = \{x_i, y_j\}, \quad f_{i,j} = f(\mathbf{x}_{i,j}).$$

Regular Grids in Two Dimensions

- A **separable integral** can be done by doing integration along one axes first, then another:

$$J = \int_{x=0}^{1} \int_{y=0}^{1} f(x,y)dxdy = \int_{x=0}^{1} dx \left[\int_{y=0}^{1} f(x,y)dy \right]$$

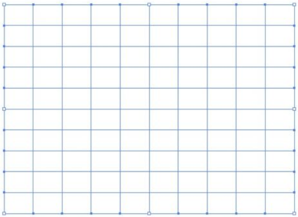
- Consider evaluating the function at nodes on a **regular grid**

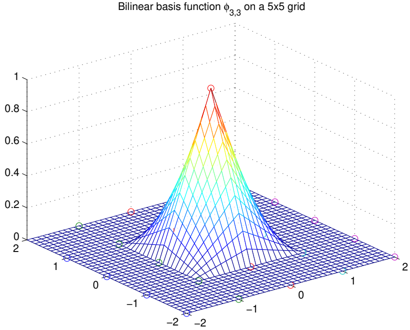
$$\mathbf{x}_{i,j} = \{x_i, y_j\}, \quad f_{i,j} = f(\mathbf{x}_{i,j}).$$

- We can use **separable basis** functions:

$$\phi_{i,j}(\mathbf{x}) = \phi_i(x)\phi_j(y).$$

Bilinear basis functions





Piecewise-Polynomial Integration

- Use a different interpolation function $\phi_{(i,j)} : \Omega_{i,j} \rightarrow \mathbb{R}$ in each rectange of the grid

$$\Omega_{i,j} = [x_i, x_{i+1}] \times [y_j, y_{j+1}],$$

and it is sufficient to look at a **unit reference rectangle** $\hat{\Omega} = [0, 1] \times [0, 1]$.

Piecewise-Polynomial Integration

- Use a different interpolation function $\phi_{(i,j)} : \Omega_{i,j} \to \mathbb{R}$ in each rectange of the grid

$$\Omega_{i,j} = [x_i, x_{i+1}] \times [y_j, y_{j+1}],$$

and it is sufficient to look at a **unit reference rectangle** $\hat{\Omega} = [0,1] \times [0,1]$.

- Recall: The equivalent of piecewise linear interpolation in 1D is the **piecewise bilinear interpolation**

Piecewise-Polynomial Integration

- Use a different interpolation function $\phi_{(i,j)} : \Omega_{i,j} \to \mathbb{R}$ in each rectange of the grid

$$\Omega_{i,j} = [x_i, x_{i+1}] \times [y_j, y_{j+1}],$$

and it is sufficient to look at a **unit reference rectangle** $\hat{\Omega} = [0,1] \times [0,1]$.

- Recall: The equivalent of piecewise linear interpolation in 1D is the **piecewise bilinear interpolation**

$$\phi_{(i,j)}(x,y) = \phi_{(i)}^{(x)}(x) \cdot \phi_{(j)}^{(y)}(y),$$

where $\phi_{(i)}^{(x)}$ and $\phi_{(j)}^{(y)}$ are linear function.

Piecewise-Polynomial Integration

- Use a different interpolation function $\phi_{(i,j)} : \Omega_{i,j} \to \mathbb{R}$ in each rectange of the grid

$$\Omega_{i,j} = [x_i, x_{i+1}] \times [y_j, y_{j+1}],$$

and it is sufficient to look at a **unit reference rectangle** $\hat{\Omega} = [0,1] \times [0,1]$.

- Recall: The equivalent of piecewise linear interpolation in 1D is the **piecewise bilinear interpolation**

$$\phi_{(i,j)}(x,y) = \phi_{(i)}^{(x)}(x) \cdot \phi_{(j)}^{(y)}(y),$$

where $\phi_{(i)}^{(x)}$ and $\phi_{(j)}^{(y)}$ are linear function.

- The global interpolant can be written in the **tent-function basis**

$$\phi(x,y) = \sum_{i,j} f_{i,j} \phi_{i,j}(x,y).$$

Bilinear Integration

- The composite **two-dimensional trapezoidal quadrature** is then:

$$J \approx \int_{x=0}^{1} \int_{y=0}^{1} \phi(x,y)dxdy = \sum_{i,j} f_{i,j} \int \int \phi_{i,j}(x,y)dxdy = \sum_{i,j} w_{i,j} f_{i,j}$$

Bilinear Integration

- The composite **two-dimensional trapezoidal quadrature** is then:

$$J \approx \int_{x=0}^{1} \int_{y=0}^{1} \phi(x,y)dxdy = \sum_{i,j} f_{i,j} \int \int \phi_{i,j}(x,y)dxdy = \sum_{i,j} w_{i,j} f_{i,j}$$

- Consider one of the corners $(0,0)$ of the reference rectangle and the corresponding basis $\hat{\phi}_{0,0}$ restricted to $\hat{\Omega}$:

Bilinear Integration

- The composite **two-dimensional trapezoidal quadrature** is then:

$$J \approx \int_{x=0}^{1} \int_{y=0}^{1} \phi(x,y)dxdy = \sum_{i,j} f_{i,j} \int \int \phi_{i,j}(x,y)dxdy = \sum_{i,j} w_{i,j} f_{i,j}$$

- Consider one of the corners $(0,0)$ of the reference rectangle and the corresponding basis $\hat{\phi}_{0,0}$ restricted to $\hat{\Omega}$:

$$\hat{\phi}_{0,0}(\hat{x},\hat{y}) = (1-\hat{x})(1-\hat{y})$$

Bilinear Integration

- The composite **two-dimensional trapezoidal quadrature** is then:

$$J \approx \int_{x=0}^{1} \int_{y=0}^{1} \phi(x,y)dxdy = \sum_{i,j} f_{i,j} \int \int \phi_{i,j}(x,y)dxdy = \sum_{i,j} w_{i,j} f_{i,j}$$

- Consider one of the corners $(0,0)$ of the reference rectangle and the corresponding basis $\hat{\phi}_{0,0}$ restricted to $\hat{\Omega}$:

$$\hat{\phi}_{0,0}(\hat{x},\hat{y}) = (1-\hat{x})(1-\hat{y})$$

- Now integrate $\hat{\phi}_{0,0}$ over $\hat{\Omega}$:

$$\int_{\hat{\Omega}} \hat{\phi}_{0,0}(\hat{x},\hat{y})d\hat{x}d\hat{y} = \frac{1}{4}.$$

Bilinear Integration

- The composite **two-dimensional trapezoidal quadrature** is then:

$$J \approx \int_{x=0}^{1} \int_{y=0}^{1} \phi(x,y)dxdy = \sum_{i,j} f_{i,j} \int \int \phi_{i,j}(x,y)dxdy = \sum_{i,j} w_{i,j} f_{i,j}$$

- Consider one of the corners $(0,0)$ of the reference rectangle and the corresponding basis $\hat{\phi}_{0,0}$ restricted to $\hat{\Omega}$:

$$\hat{\phi}_{0,0}(\hat{x},\hat{y}) = (1-\hat{x})(1-\hat{y})$$

- Now integrate $\hat{\phi}_{0,0}$ over $\hat{\Omega}$:

$$\int_{\hat{\Omega}} \hat{\phi}_{0,0}(\hat{x},\hat{y})d\hat{x}d\hat{y} = \frac{1}{4}.$$

- Since each **interior node** contributes to 4 rectangles, its weight is 1.

Bilinear Integration

- The composite **two-dimensional trapezoidal quadrature** is then:

$$J \approx \int_{x=0}^{1} \int_{y=0}^{1} \phi(x,y)dxdy = \sum_{i,j} f_{i,j} \int \int \phi_{i,j}(x,y)dxdy = \sum_{i,j} w_{i,j} f_{i,j}$$

- Consider one of the corners $(0,0)$ of the reference rectangle and the corresponding basis $\hat{\phi}_{0,0}$ restricted to $\hat{\Omega}$:

$$\hat{\phi}_{0,0}(\hat{x},\hat{y}) = (1-\hat{x})(1-\hat{y})$$

- Now integrate $\hat{\phi}_{0,0}$ over $\hat{\Omega}$:

$$\int_{\hat{\Omega}} \hat{\phi}_{0,0}(\hat{x},\hat{y})d\hat{x}d\hat{y} = \frac{1}{4}.$$

- Since each **interior node** contributes to 4 rectangles, its weight is 1. **Edge nodes** contribute to 2 rectangles, so their weight is $1/2$.

Bilinear Integration

- The composite **two-dimensional trapezoidal quadrature** is then:

$$J \approx \int_{x=0}^{1} \int_{y=0}^{1} \phi(x,y)dxdy = \sum_{i,j} f_{i,j} \int \int \phi_{i,j}(x,y)dxdy = \sum_{i,j} w_{i,j} f_{i,j}$$

- Consider one of the corners $(0,0)$ of the reference rectangle and the corresponding basis $\hat{\phi}_{0,0}$ restricted to $\hat{\Omega}$:

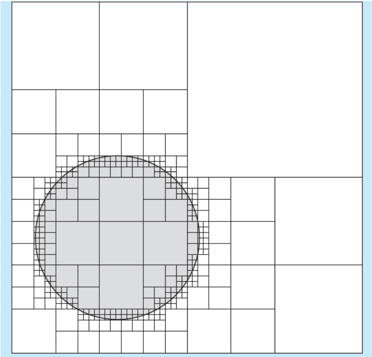
$$\hat{\phi}_{0,0}(\hat{x},\hat{y}) = (1-\hat{x})(1-\hat{y})$$

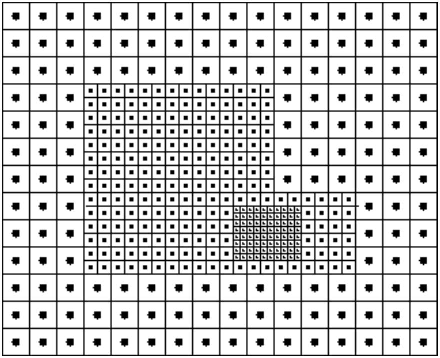
- Now integrate $\hat{\phi}_{0,0}$ over $\hat{\Omega}$:

$$\int_{\hat{\Omega}} \hat{\phi}_{0,0}(\hat{x},\hat{y})d\hat{x}d\hat{y} = \frac{1}{4}.$$

- Since each **interior node** contributes to 4 rectangles, its weight is 1. **Edge nodes** contribute to 2 rectangles, so their weight is $1/2$. **Corners** contribute to only one rectangle, so their weight is $1/4$.

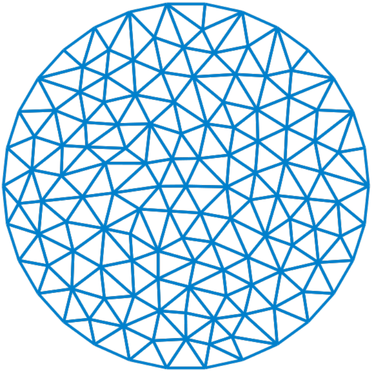
Adaptive Meshes: Quadtrees and Block-Structured

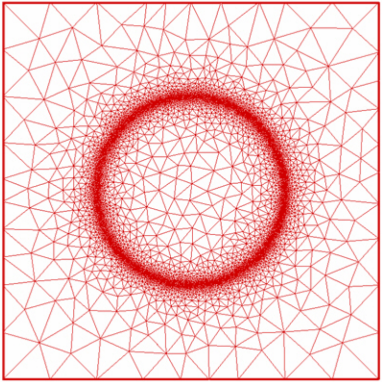




Irregular (Simplicial) Meshes

Any polygon can be triangulated into arbitrarily many **disjoint triangles**. Similarly **tetrahedral meshes** in 3D.





Outline

In MATLAB

- The MATLAB function $quad(f, a, b, \varepsilon)$ uses adaptive Simpson quadrature to compute the integral.

In MATLAB

- The MATLAB function $quad(f, a, b, \varepsilon)$ uses adaptive Simpson quadrature to compute the integral.
- The MATLAB function $quadl(f, a, b, \varepsilon)$ uses adaptive Gauss-Lobatto quadrature.

In MATLAB

- The MATLAB function $quad(f, a, b, \varepsilon)$ uses adaptive Simpson quadrature to compute the integral.
- The MATLAB function $quadl(f, a, b, \varepsilon)$ uses adaptive Gauss-Lobatto quadrature.
- MATLAB says: "The function *quad* may be more efficient with low accuracies or nonsmooth integrands."

In MATLAB

- The MATLAB function $quad(f, a, b, \varepsilon)$ uses adaptive Simpson quadrature to compute the integral.
- The MATLAB function $quadl(f, a, b, \varepsilon)$ uses adaptive Gauss-Lobatto quadrature.
- MATLAB says: "The function *quad* may be more efficient with low accuracies or nonsmooth integrands."
- In two dimensions, for separable integrals over rectangles, use

$$J = dblquad(f, x_{min}, x_{max}, y_{min}, y_{max}, \varepsilon)$$

$$J = dblquad(f, x_{min}, x_{max}, y_{min}, y_{max}, \varepsilon, @quadl)$$

In MATLAB

- The MATLAB function $quad(f, a, b, \varepsilon)$ uses adaptive Simpson quadrature to compute the integral.
- The MATLAB function $quadl(f, a, b, \varepsilon)$ uses adaptive Gauss-Lobatto quadrature.
- MATLAB says: "The function *quad* may be more efficient with low accuracies or nonsmooth integrands."
- In two dimensions, for separable integrals over rectangles, use

$$J = dblquad(f, x_{min}, x_{max}, y_{min}, y_{max}, \varepsilon)$$

$$J = dblquad(f, x_{min}, x_{max}, y_{min}, y_{max}, \varepsilon, @quadl)$$

- There is also *triplequad*.

Conclusions/Summary

- Numerical integration or quadrature approximates an integral via a discrete **weighted sum** of function values over a set of **nodes**.

Conclusions/Summary

- Numerical integration or quadrature approximates an integral via a discrete **weighted sum** of function values over a set of **nodes**.
- Integration is based on interpolation: Integrate the interpolant to get a good approximation.

Conclusions/Summary

- Numerical integration or quadrature approximates an integral via a discrete **weighted sum** of function values over a set of **nodes**.
- Integration is based on interpolation: Integrate the interpolant to get a good approximation.
- Piecewise polynomial interpolation over **equi-spaced nodes** gives the **trapezoidal and Simpson quadratures** for lower order, and higher order are generally not recommended.

Conclusions/Summary

- Numerical integration or quadrature approximates an integral via a discrete **weighted sum** of function values over a set of **nodes**.
- Integration is based on interpolation: Integrate the interpolant to get a good approximation.
- Piecewise polynomial interpolation over **equi-spaced nodes** gives the **trapezoidal and Simpson quadratures** for lower order, and higher order are generally not recommended.
- In higher dimensions we split the domain into **rectangles for regular grids** (separable integration), or **triangles/tetrahedra for simplicial meshes**.

Conclusions/Summary

- Numerical integration or quadrature approximates an integral via a discrete **weighted sum** of function values over a set of **nodes**.
- Integration is based on interpolation: Integrate the interpolant to get a good approximation.
- Piecewise polynomial interpolation over **equi-spaced nodes** gives the **trapezoidal and Simpson quadratures** for lower order, and higher order are generally not recommended.
- In higher dimensions we split the domain into **rectangles for regular grids** (separable integration), or **triangles/tetrahedra for simplicial meshes**.
- Integration in high dimensions d becomes harder and harder because the number of nodes grows as N^d: **Curse of dimensionality**. Monte Carlo is one possible cure...