MATH 350: Introduction to Computational Mathematics

Chapter VI: Numerical Integration

Greg Fasshauer

Department of Applied Mathematics
Illinois Institute of Technology

Spring 2011

Outline

Outline

Numerical integration — the process of evaluating a definite integral numerically — is also known as quadrature.

Numerical integration — the process of evaluating a definite integral numerically — is also known as quadrature.

Certain integrals of “simple” functions can not be evaluated by basic analytical (i.e., calculus) methods.

Numerical integration — the process of evaluating a definite integral numerically — is also known as quadrature.
Certain integrals of "simple" functions can not be evaluated by basic analytical (i.e., calculus) methods.

Example

Such examples include

- the normal distribution function

$$\int_0^x e^{-t^2} dt$$

Numerical integration — the process of evaluating a definite integral numerically — is also known as quadrature.

Certain integrals of “simple” functions can not be evaluated by basic analytical (i.e., calculus) methods.

Example

Such examples include

- the normal distribution function

$$\int_0^x e^{-t^2} dt = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x),$$

Numerical integration — the process of evaluating a definite integral numerically — is also known as quadrature.
Certain integrals of "simple" functions can not be evaluated by basic analytical (i.e., calculus) methods.

Example

Such examples include

- the normal distribution function

$$\int_0^x e^{-t^2} dt = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x),$$

- arc length calculations such as

$$\int_{-1}^{1} \sqrt{1+x^3} dx$$

Numerical integration — the process of evaluating a definite integral numerically — is also known as quadrature.
Certain integrals of “simple” functions can not be evaluated by basic analytical (i.e., calculus) methods.

Example

Such examples include

- the normal distribution function

$$\int_0^x e^{-t^2} dt = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x),$$

- arc length calculations such as

$$\int_{-1}^1 \sqrt{1+x^3} dx = \frac{2\sqrt{2}}{5} + \frac{3^{3/4}}{5} \left[2K\left(\frac{\sqrt{2}+\sqrt{6}}{4}\right) - F\left(\frac{2\sqrt{2}3^{1/4}}{2+\sqrt{3}}, \frac{\sqrt{2}+\sqrt{6}}{4}\right) \right],$$

  where K and F are complete and incomplete elliptic integrals of the first kind, respectively,

Example (cont.)

- a “harmless” trigonometric integral

$$\int_0^1 \cos(x^3)\mathrm{d}x$$

Example (cont.)

- a “harmless” trigonometric integral

$$\int_0^1 \cos(x^3)\mathrm{d}x = \tfrac{3}{7}\sin(1) + \cos(1) - \tfrac{3}{7}\sin(1)s_{\frac{11}{6},\frac{3}{2}}(1) - s_{\frac{5}{6},\frac{1}{2}}(1)\left(\cos(1) - \sin(1)\right),$$

with the Lommel function $s_{\alpha,\beta}$.

Example (cont.)

- a “harmless” trigonometric integral

$$\int_0^1 \cos(x^3)\mathrm{d}x = \tfrac{3}{7}\sin(1) + \cos(1) - \tfrac{3}{7}\sin(1)s_{\frac{11}{6},\frac{3}{2}}(1) - s_{\frac{5}{6},\frac{1}{2}}(1)\left(\cos(1) - \sin(1)\right),$$

  with the Lommel function $s_{\alpha,\beta}$.

- the Bessel function J_0

$$\frac{1}{\pi}\int_0^\pi \cos(x\sin t)\mathrm{d}t = J_0(x),$$

Example (cont.)

- a "harmless" trigonometric integral

$$\int_0^1 \cos(x^3)\mathrm{d}x = \tfrac{3}{7}\sin(1) + \cos(1) - \tfrac{3}{7}\sin(1)s_{\frac{11}{6},\frac{3}{2}}(1) - s_{\frac{5}{6},\frac{1}{2}}(1)\left(\cos(1) - \sin(1)\right),$$

  with the Lommel function $s_{\alpha,\beta}$.

- the Bessel function J_0

$$\frac{1}{\pi}\int_0^\pi \cos(x\sin t)\mathrm{d}t = J_0(x),$$

- a complicated integral involving J_0 that even the current versions of Maple or Mathematica/Alpha can't handle analytically

$$\int_0^1 J_0(x)xe^{x^2}\mathrm{d}x,$$

Example (cont.)

- the sine integral

$$\int_0^x \frac{\sin t}{t} \mathrm{d}t = \mathrm{Si}(x),$$

Example (cont.)

- the sine integral

$$\int_0^x \frac{\sin t}{t} \mathrm{d}t = \mathrm{Si}(x),$$

- or problems from engineering such as this integral which plays a role in Debye's model for calculating the heat capacity of a solid:

$$\int_0^x \frac{t^3}{e^t - 1} \mathrm{d}t$$

Example (cont.)

- the sine integral

$$\int_0^x \frac{\sin t}{t} \mathrm{d}t = \mathrm{Si}(x),$$

- or problems from engineering such as this integral which plays a role in Debye's model for calculating the heat capacity of a solid:

$$\begin{aligned}\int_0^x \frac{t^3}{\mathrm{e}^t - 1} \mathrm{d}t &= -\tfrac{\pi^4}{15} - \tfrac{x^4}{4} + x^3 \ln(1 - \mathrm{e}^x) + 3x^2 \mathrm{Li}_2(\mathrm{e}^x) \\ &\quad -6x\mathrm{Li}_3(\mathrm{e}^x) + 6\mathrm{Li}_4(\mathrm{e}^x).\end{aligned}$$

Here Li_n is the polylogarithm of index n.

Example (cont.)

- the sine integral

$$\int_0^x \frac{\sin t}{t} \mathrm{d}t = \mathrm{Si}(x),$$

- or problems from engineering such as this integral which plays a role in Debye's model for calculating the heat capacity of a solid:

$$\int_0^x \frac{t^3}{e^t - 1} \mathrm{d}t = -\frac{\pi^4}{15} - \frac{x^4}{4} + x^3 \ln(1 - e^x) + 3x^2 \mathrm{Li}_2(e^x) - 6x\mathrm{Li}_3(e^x) + 6\mathrm{Li}_4(e^x).$$

Here Li_n is the polylogarithm of index n.

Other "functions" are not even given in closed form, but only as a set of discrete values (for example as measurements in an experiments, or as output from another computer simulation).

Numerical integration is also closely related to the solution of differential equations.

Numerical integration is also closely related to the solution of differential equations.

Example

Solve the simple first-order differential equation $y'(t) = ky(t)$.

Numerical integration is also closely related to the solution of differential equations.

Example

Solve the simple first-order differential equation $y'(t) = ky(t)$.
Clearly, this can be achieved by separation and integration:

$$y' = ky$$

Numerical integration is also closely related to the solution of differential equations.

Example

Solve the simple first-order differential equation $y'(t) = ky(t)$.
Clearly, this can be achieved by separation and integration:

$$y' = ky \iff \frac{\mathrm{d}y}{\mathrm{d}t} = ky$$

Numerical integration is also closely related to the solution of differential equations.

Example

Solve the simple first-order differential equation $y'(t) = ky(t)$.
Clearly, this can be achieved by separation and integration:

$$y' = ky \iff \frac{\mathrm{d}y}{\mathrm{d}t} = ky \iff \frac{\mathrm{d}y}{y} = k\mathrm{d}t$$

Numerical integration is also closely related to the solution of differential equations.

Example

Solve the simple first-order differential equation $y'(t) = ky(t)$. Clearly, this can be achieved by separation and integration:

$$y' = ky \iff \frac{\mathrm{d}y}{\mathrm{d}t} = ky \iff \frac{\mathrm{d}y}{y} = k\mathrm{d}t$$

$$\iff \int \frac{\mathrm{d}y}{y} = \int k\mathrm{d}t$$

Numerical integration is also closely related to the solution of differential equations.

Example

Solve the simple first-order differential equation $y'(t) = ky(t)$. Clearly, this can be achieved by separation and integration:

$$y' = ky \iff \frac{\mathrm{d}y}{\mathrm{d}t} = ky \iff \frac{\mathrm{d}y}{y} = k\mathrm{d}t$$

$$\iff \int \frac{\mathrm{d}y}{y} = \int k\mathrm{d}t \iff \ln|y| = kt + c$$

Numerical integration is also closely related to the solution of differential equations.

Example

Solve the simple first-order differential equation $y'(t) = ky(t)$.
Clearly, this can be achieved by separation and integration:

$$y' = ky \iff \frac{\mathrm{d}y}{\mathrm{d}t} = ky \iff \frac{\mathrm{d}y}{y} = k\mathrm{d}t$$
$$\iff \int \frac{\mathrm{d}y}{y} = \int k\mathrm{d}t \iff \ln|y| = kt + c \iff y = Ce^{kt}.$$

Numerical integration is also closely related to the solution of differential equations.

Example

Solve the simple first-order differential equation $y'(t) = ky(t)$.
Clearly, this can be achieved by separation and integration:

$$y' = ky \iff \frac{\mathrm{d}y}{\mathrm{d}t} = ky \iff \frac{\mathrm{d}y}{y} = k\mathrm{d}t$$

$$\iff \int \frac{\mathrm{d}y}{y} = \int k\mathrm{d}t \iff \ln|y| = kt + c \iff y = Ce^{kt}.$$

However, here no numerical integration was required.

Example

More generally, an **arbitrary** first-order differential equation

$$y'(t) = f(t, y(t)) \tag{1}$$

can also be solved by integration

Example

More generally, an **arbitrary** first-order differential equation

$$y'(t) = f(t, y(t)) \tag{1}$$

can also be solved by integration, i.e., we integrate (1) on a (small) interval $[t, t+h]$ to get

$$\int_t^{t+h} y'(\tau)\mathrm{d}\tau = y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau. \tag{2}$$

Example

More generally, an **arbitrary** first-order differential equation

$$y'(t) = f(t, y(t)) \tag{1}$$

can also be solved by integration, i.e., we integrate (1) on a (small) interval $[t, t+h]$ to get

$$\int_t^{t+h} y'(\tau)\mathrm{d}\tau = y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau. \tag{2}$$

If we assume that f is almost constant on $[t, t+h]$, i.e., $f(\tau, y(\tau)) \approx f(t, y(t))$ for $\tau \in [t, t+h]$, then we further have

$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx \int_t^{t+h} f(t, y(t))\mathrm{d}\tau$$

Example

More generally, an **arbitrary** first-order differential equation

$$y'(t) = f(t, y(t)) \tag{1}$$

can also be solved by integration, i.e., we integrate (1) on a (small) interval $[t, t+h]$ to get

$$\int_t^{t+h} y'(\tau)\mathrm{d}\tau = y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau. \tag{2}$$

If we assume that f is almost constant on $[t, t+h]$, i.e., $f(\tau, y(\tau)) \approx f(t, y(t))$ for $\tau \in [t, t+h]$, then we further have

$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx \int_t^{t+h} f(t, y(t))\mathrm{d}\tau = f(t, y(t)) \int_t^{t+h} \mathrm{d}\tau$$

Example

More generally, an **arbitrary** first-order differential equation

$$y'(t) = f(t, y(t)) \tag{1}$$

can also be solved by integration, i.e., we integrate (1) on a (small) interval $[t, t+h]$ to get

$$\int_t^{t+h} y'(\tau)\mathrm{d}\tau = y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau. \tag{2}$$

If we assume that f is almost constant on $[t, t+h]$, i.e., $f(\tau, y(\tau)) \approx f(t, y(t))$ for $\tau \in [t, t+h]$, then we further have

$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx \int_t^{t+h} f(t, y(t))\mathrm{d}\tau = f(t, y(t)) \int_t^{t+h} \mathrm{d}\tau = f(t, y(t))h.$$

Example

More generally, an **arbitrary** first-order differential equation

$$y'(t) = f(t, y(t)) \tag{1}$$

can also be solved by integration, i.e., we integrate (1) on a (small) interval $[t, t+h]$ to get

$$\int_t^{t+h} y'(\tau)\mathrm{d}\tau = y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau. \tag{2}$$

If we assume that f is almost constant on $[t, t+h]$, i.e., $f(\tau, y(\tau)) \approx f(t, y(t))$ for $\tau \in [t, t+h]$, then we further have

$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx \int_t^{t+h} f(t, y(t))\mathrm{d}\tau = f(t, y(t)) \int_t^{t+h} \mathrm{d}\tau = f(t, y(t))h.$$

Combining this with (2), we end up with Euler's method from Chapter 1:

$$y(t+h) \approx y(t) + hf(t, y(t)).$$

The general idea

Any numerical integration/quadrature method will replace a given (continuous) integral by a (discrete) sum, i.e.,

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^n w_i f(x_i),$$

where the w_i are weights and the x_i are integration nodes both of which characterize a specific rule.

The general idea

Any numerical integration/quadrature method will replace a given (continuous) integral by a (discrete) sum, i.e.,

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i),$$

where the w_i are weights and the x_i are integration nodes both of which characterize a specific rule.

Note that for the integration leading to Euler's method we have

The general idea

Any numerical integration/quadrature method will replace a given (continuous) integral by a (discrete) sum, i.e.,

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i),$$

where the w_i are weights and the x_i are integration nodes both of which characterize a specific rule.

Note that for the integration leading to Euler's method we have

$$a = t, \quad b = t + h, \quad n = 1, \quad w_1 = h = \frac{b-a}{n} \quad \text{and} \quad x_1 = t.$$

The general idea

Any numerical integration/quadrature method will replace a given (continuous) integral by a (discrete) sum, i.e.,

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i),$$

where the w_i are weights and the x_i are integration nodes both of which characterize a specific rule.

Note that for the integration leading to Euler's method we have

$$a = t, \quad b = t + h, \quad n = 1, \quad w_1 = h = \frac{b-a}{n} \quad \text{and} \quad x_1 = t.$$

Since we approximate an integral by a sum, using a quadrature rule will generally come at the expense of a truncation error.

Example

Monte Carlo integration assumes that all weights are equal, i.e., $w_i = \frac{b-a}{n}$, and the integration nodes are random points in the interval $[a, b]$.

Example

Monte Carlo integration assumes that all weights are equal, i.e., $w_i = \frac{b-a}{n}$, and the integration nodes are random points in the interval $[a, b]$.

Then

$$\frac{1}{b-a}\int_a^b f(x)\mathrm{d}x \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i),$$

Example

Monte Carlo integration assumes that all weights are equal, i.e., $w_i = \frac{b-a}{n}$, and the integration nodes are random points in the interval $[a, b]$.
Then

$$\frac{1}{b-a} \int_a^b f(x) \mathrm{d}x \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i),$$

i.e., we approximate the average of the function by the average of a random set of function values.

Example

Monte Carlo integration assumes that all weights are equal, i.e., $w_i = \frac{b-a}{n}$, and the integration nodes are random points in the interval $[a, b]$.
Then

$$\frac{1}{b-a} \int_a^b f(x) \mathrm{d}x \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i),$$

i.e., we approximate the average of the function by the average of a random set of function values.

This method is extremely simple to use and one of the few viable methods for high-dimensional integrals.

On the downside, it is not very accurate, i.e., n may have to be very large to get an acceptable result.

Outline

In Calculus the Riemann integral

$$\int_a^b f(x)\mathrm{d}x = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x$$

is defined as the limit of the sum of areas of ever-thinner rectangles.

In Calculus the Riemann integral

$$\int_a^b f(x)\mathrm{d}x = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x$$

is defined as the limit of the sum of areas of ever-thinner rectangles. By dropping the limit, i.e., taking only a finite number, n, of terms in the sum we get a numerical integration method:

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} f(x_i^*)\Delta x$$

In Calculus the Riemann integral

$$\int_a^b f(x)\mathrm{d}x = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x$$

is defined as the limit of the sum of areas of ever-thinner rectangles.

By dropping the limit, i.e., taking only a finite number, n, of terms in the sum we get a numerical integration method:

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} f(x_i^*)\Delta x$$

Depending on how we select the points x_i^*, we will obtain slightly different rules:

- left endpoint rule: x_i^* at the left endpoint of each sub-interval,
- right endpoint rule: x_i^* at the right endpoint of each sub-interval,
- midpoint rule: x_i^* at the midpoint of each sub-interval.

In Calculus the Riemann integral

$$\int_a^b f(x)\mathrm{d}x = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x$$

is defined as the limit of the sum of areas of ever-thinner rectangles.
By dropping the limit, i.e., taking only a finite number, n, of terms in the sum we get a numerical integration method:

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} f(x_i^*)\Delta x$$

Depending on how we select the points x_i^*, we will obtain slightly different rules:

- left endpoint rule: x_i^* at the left endpoint of each sub-interval,
- right endpoint rule: x_i^* at the right endpoint of each sub-interval,
- midpoint rule: x_i^* at the midpoint of each sub-interval.

See the Maple worksheet `Integration.mw` or Maple's *Approximate Integration Tutor*.

Formulas

Split $[a, b]$ into n subintervals (each of length $h = \frac{b-a}{n}$), and let $x_i = a + ih$, $i = 0, \ldots, n$, be the endpoints of these subintervals.

- Left endpoint rule:

$$\int_a^b f(x)\mathrm{d}x \approx L_n(f) = \sum_{i=1}^{n} hf(x_{i-1}),$$

  i.e., weights are all equal, $w_i = h$, and nodes are left endpoints.

- Right endpoint rule analogous:

$$\int_a^b f(x)\mathrm{d}x \approx R_n(f) = \sum_{i=1}^{n} hf(x_i).$$

- Midpoint rule:

$$\int_a^b f(x)\mathrm{d}x \approx M_n(f) = \sum_{i=1}^{n} hf\left(\frac{x_{i-1} + x_i}{2}\right).$$

  Again all equal weights, but nodes at midpoints.

Truncation errors

Perform the convergence experiments in `Integration.mw`.

Truncation errors

Perform the convergence experiments in `Integration.mw`.

- Left endpoint rule:

$$\int_a^b f(x)\mathrm{d}x = L_n(f) + \mathcal{O}(h).$$

- Right endpoint rule:

$$\int_a^b f(x)\mathrm{d}x = R_n(f) + \mathcal{O}(h).$$

- Midpoint rule:

$$\int_a^b f(x)\mathrm{d}x = M_n(f) + \mathcal{O}(h^2).$$

Truncation errors

Perform the convergence experiments in `Integration.mw`.

- Left endpoint rule:

$$\int_a^b f(x)\mathrm{d}x = L_n(f) + \mathcal{O}(h).$$

- Right endpoint rule:

$$\int_a^b f(x)\mathrm{d}x = R_n(f) + \mathcal{O}(h).$$

- Midpoint rule:

$$\int_a^b f(x)\mathrm{d}x = M_n(f) + \mathcal{O}(h^2).$$

Remark

While all methods converge to the Riemann integral limit, they do this at different rates! The midpoint rule is generally much more accurate — even though all three methods approximate the integrand by a constant on each subinterval.

With the simplest rules from the previous slides we approximate the integrand by a constant on each subinterval.

With the simplest rules from the previous slides we approximate the integrand by a constant on each subinterval.

We can do better by using polynomial interpolants of f.

With the simplest rules from the previous slides we approximate the integrand by a constant on each subinterval.

We can do better by using polynomial interpolants of f.

Use of a linear interpolant on each subinterval $[x_{i-1}, x_i]$, $i = 1, \ldots, n$, leads to the trapezoidal rule, i.e.,

$$\int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \approx \int_{x_{i-1}}^{x_i} p(x)\mathrm{d}x,$$

where

$$p(x) = \frac{x - x_i}{x_{i-1} - x_i} f(x_{i-1}) + \frac{x - x_{i-1}}{x_i - x_{i-1}} f(x_i).$$

Derivation of Trapezoid Rule

$$\int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \approx \int_{x_{i-1}}^{x_i} \left[\frac{x - x_i}{x_{i-1} - x_i} f(x_{i-1}) + \frac{x - x_{i-1}}{x_i - x_{i-1}} f(x_i)\right] \mathrm{d}x$$

Derivation of Trapezoid Rule

$$\begin{aligned}
\int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x &\approx \int_{x_{i-1}}^{x_i} \left[\frac{x - x_i}{x_{i-1} - x_i} f(x_{i-1}) + \frac{x - x_{i-1}}{x_i - x_{i-1}} f(x_i)\right] \mathrm{d}x \\
&= \frac{f(x_{i-1})}{x_{i-1} - x_i} \int_{x_{i-1}}^{x_i} (x - x_i)\mathrm{d}x + \frac{f(x_i)}{x_i - x_{i-1}} \int_{x_{i-1}}^{x_i} (x - x_{i-1})\mathrm{d}x
\end{aligned}$$

Derivation of Trapezoid Rule

$$\int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \approx \int_{x_{i-1}}^{x_i} \left[\frac{x - x_i}{x_{i-1} - x_i} f(x_{i-1}) + \frac{x - x_{i-1}}{x_i - x_{i-1}} f(x_i)\right] \mathrm{d}x$$

$$= \frac{f(x_{i-1})}{x_{i-1} - x_i} \int_{x_{i-1}}^{x_i} (x - x_i)\mathrm{d}x + \frac{f(x_i)}{x_i - x_{i-1}} \int_{x_{i-1}}^{x_i} (x - x_{i-1})\mathrm{d}x$$

$$= \frac{f(x_{i-1})}{x_{i-1} - x_i} \left[\frac{(x - x_i)^2}{2}\right]_{x_{i-1}}^{x_i} + \frac{f(x_i)}{x_i - x_{i-1}} \left[\frac{(x - x_{i-1})^2}{2}\right]_{x_{i-1}}^{x_i}$$

Derivation of Trapezoid Rule

$$
\begin{aligned}
\int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x &\approx \int_{x_{i-1}}^{x_i} \left[\frac{x - x_i}{x_{i-1} - x_i} f(x_{i-1}) + \frac{x - x_{i-1}}{x_i - x_{i-1}} f(x_i)\right] \mathrm{d}x \\
&= \frac{f(x_{i-1})}{x_{i-1} - x_i} \int_{x_{i-1}}^{x_i} (x - x_i)\mathrm{d}x + \frac{f(x_i)}{x_i - x_{i-1}} \int_{x_{i-1}}^{x_i} (x - x_{i-1})\mathrm{d}x \\
&= \frac{f(x_{i-1})}{x_{i-1} - x_i} \left[\frac{(x - x_i)^2}{2}\right]_{x_{i-1}}^{x_i} + \frac{f(x_i)}{x_i - x_{i-1}} \left[\frac{(x - x_{i-1})^2}{2}\right]_{x_{i-1}}^{x_i} \\
&= -\frac{f(x_{i-1})}{x_{i-1} - x_i} \frac{(x_{i-1} - x_i)^2}{2} + \frac{f(x_i)}{x_i - x_{i-1}} \frac{(x_i - x_{i-1})^2}{2}
\end{aligned}
$$

Derivation of Trapezoid Rule

$$
\begin{aligned}
\int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x &\approx \int_{x_{i-1}}^{x_i} \left[\frac{x-x_i}{x_{i-1}-x_i} f(x_{i-1}) + \frac{x-x_{i-1}}{x_i-x_{i-1}} f(x_i)\right] \mathrm{d}x \\
&= \frac{f(x_{i-1})}{x_{i-1}-x_i} \int_{x_{i-1}}^{x_i} (x-x_i)\mathrm{d}x + \frac{f(x_i)}{x_i-x_{i-1}} \int_{x_{i-1}}^{x_i} (x-x_{i-1})\mathrm{d}x \\
&= \frac{f(x_{i-1})}{x_{i-1}-x_i} \left[\frac{(x-x_i)^2}{2}\right]_{x_{i-1}}^{x_i} + \frac{f(x_i)}{x_i-x_{i-1}} \left[\frac{(x-x_{i-1})^2}{2}\right]_{x_{i-1}}^{x_i} \\
&= -\frac{f(x_{i-1})}{x_{i-1}-x_i} \frac{(x_{i-1}-x_i)^2}{2} + \frac{f(x_i)}{x_i-x_{i-1}} \frac{(x_i-x_{i-1})^2}{2} \\
&= f(x_{i-1}) \frac{x_i-x_{i-1}}{2} + f(x_i) \frac{x_i-x_{i-1}}{2}
\end{aligned}
$$

Derivation of Trapezoid Rule

$$
\begin{aligned}
\int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x &\approx \int_{x_{i-1}}^{x_i} \left[\frac{x - x_i}{x_{i-1} - x_i} f(x_{i-1}) + \frac{x - x_{i-1}}{x_i - x_{i-1}} f(x_i)\right] \mathrm{d}x \\
&= \frac{f(x_{i-1})}{x_{i-1} - x_i} \int_{x_{i-1}}^{x_i} (x - x_i)\mathrm{d}x + \frac{f(x_i)}{x_i - x_{i-1}} \int_{x_{i-1}}^{x_i} (x - x_{i-1})\mathrm{d}x \\
&= \frac{f(x_{i-1})}{x_{i-1} - x_i} \left[\frac{(x - x_i)^2}{2}\right]_{x_{i-1}}^{x_i} + \frac{f(x_i)}{x_i - x_{i-1}} \left[\frac{(x - x_{i-1})^2}{2}\right]_{x_{i-1}}^{x_i} \\
&= -\frac{f(x_{i-1})}{x_{i-1} - x_i} \frac{(x_{i-1} - x_i)^2}{2} + \frac{f(x_i)}{x_i - x_{i-1}} \frac{(x_i - x_{i-1})^2}{2} \\
&= f(x_{i-1}) \frac{x_i - x_{i-1}}{2} + f(x_i) \frac{x_i - x_{i-1}}{2} \\
&= (x_i - x_{i-1}) \frac{f(x_{i-1}) + f(x_i)}{2}
\end{aligned}
$$

So far we've only considered one subinterval. Putting them all together we get

$$\int_a^b f(x)\mathrm{d}x = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x$$

So far we've only considered one subinterval. Putting them all together we get

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \\ &\approx \sum_{i=1}^{n}\underbrace{(x_i - x_{i-1})}_{=h}\frac{f(x_{i-1}) + f(x_i)}{2}\end{aligned}$$

So far we've only considered one subinterval. Putting them all together we get

$$\begin{aligned}
\int_a^b f(x)\mathrm{d}x &= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \\
&\approx \sum_{i=1}^{n} \underbrace{(x_i - x_{i-1})}_{=h} \frac{f(x_{i-1}) + f(x_i)}{2} \\
&= \frac{h}{2} \sum_{i=1}^{n} \left(f(x_{i-1}) + f(x_i)\right)
\end{aligned}$$

So far we've only considered one subinterval. Putting them all together we get

$$\begin{aligned}
\int_a^b f(x)\mathrm{d}x &= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \\
&\approx \sum_{i=1}^{n} \underbrace{(x_i - x_{i-1})}_{=h} \frac{f(x_{i-1}) + f(x_i)}{2} \\
&= \frac{h}{2} \sum_{i=1}^{n} \left(f(x_{i-1}) + f(x_i)\right) \\
&= \frac{h}{2} \left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n) \right] \\
&= T_n(f)
\end{aligned}$$

So far we've only considered one subinterval. Putting them all together we get

$$\begin{aligned}
\int_a^b f(x)\mathrm{d}x &= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x \\
&\approx \sum_{i=1}^{n} \underbrace{(x_i - x_{i-1})}_{=h} \frac{f(x_{i-1}) + f(x_i)}{2} \\
&= \frac{h}{2} \sum_{i=1}^{n} \left(f(x_{i-1}) + f(x_i)\right) \\
&= \frac{h}{2} \left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n) \right] \\
&= T_n(f)
\end{aligned}$$

This is known as the composite trapezoidal rule. Note that the interior weights are twice those at the endpoints.

Now we use a quadratic interpolant over two subintervals $[x_{i-1}, x_i] \cup [x_i, x_{i+1}]$. This will lead to Simpson's rule, i.e.,

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\mathrm{d}x \approx \int_{x_{i-1}}^{x_{i+1}} p(x)\mathrm{d}x,$$

where

$$\begin{aligned} p(x) &= \frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} f(x_{i-1}) + \frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} f(x_i) \\ &\quad + \frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)} f(x_{i+1}). \end{aligned}$$

We can derive (see HW) the basic Simpson's rule for just two subintervals:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\mathrm{d}x \approx \frac{h}{3}\left[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})\right],$$

where $h = x_{i+1} - x_i = x_i - x_{i-1}$.
The composite Simpson rule then turns out to be

$$\int_a^b f(x)\mathrm{d}x = \sum_{i=1}^{n/2} \int_{x_{2i-2}}^{x_{2i}} f(x)\mathrm{d}x$$

We can derive (see HW) the basic Simpson's rule for just two subintervals:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\mathrm{d}x \approx \frac{h}{3}\left[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})\right],$$

where $h = x_{i+1} - x_i = x_i - x_{i-1}$.
The composite Simpson rule then turns out to be

$$\int_a^b f(x)\mathrm{d}x = \sum_{i=1}^{n/2} \int_{x_{2i-2}}^{x_{2i}} f(x)\mathrm{d}x \approx \sum_{i=1}^{n/2} \frac{h}{3}\left[f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})\right]$$

We can derive (see HW) the basic Simpson's rule for just two subintervals:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\mathrm{d}x \approx \frac{h}{3}\left[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})\right],$$

where $h = x_{i+1} - x_i = x_i - x_{i-1}$.
The composite Simpson rule then turns out to be

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= \sum_{i=1}^{n/2}\int_{x_{2i-2}}^{x_{2i}} f(x)\mathrm{d}x \approx \sum_{i=1}^{n/2}\frac{h}{3}\left[f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})\right] \\ &= \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 4f(x_{n-1}) + f(x_n)\right]\end{aligned}$$

We can derive (see HW) the basic Simpson's rule for just two subintervals:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\mathrm{d}x \approx \frac{h}{3}\left[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})\right],$$

where $h = x_{i+1} - x_i = x_i - x_{i-1}$.
The composite Simpson rule then turns out to be

$$\begin{aligned}
\int_a^b f(x)\mathrm{d}x &= \sum_{i=1}^{n/2} \int_{x_{2i-2}}^{x_{2i}} f(x)\mathrm{d}x \approx \sum_{i=1}^{n/2} \frac{h}{3}\left[f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})\right] \\
&= \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 4f(x_{n-1}) + f(x_n)\right] \\
&= \frac{h}{3}\left[f(x_0) + 4\sum_{i=1}^{n/2} f(x_{2i-1}) + 2\sum_{i=1}^{n/2-1} f(x_{2i}) + f(x_n)\right] \\
&= S_n(f),
\end{aligned}$$

where $h = \frac{b-a}{n}$. Note that for Simpson's rule n has to be even.

We can also use a general interpolating polynomial of degree $n-1$ given in Lagrange form

$$p(x) = \sum_{k=1}^{n} L_k(x) y_k$$

with Lagrange basis polynomials $L_k(x) = \prod_{j=1, j \neq k}^{n} \frac{x - x_j}{x_k - x_j}$, $k = 1, \ldots, n$.

We can also use a general interpolating polynomial of degree $n-1$ given in Lagrange form

$$p(x) = \sum_{k=1}^{n} L_k(x) y_k$$

with Lagrange basis polynomials $L_k(x) = \prod_{j=1, j\neq k}^{n} \frac{x-x_j}{x_k-x_j}$, $k = 1, \ldots, n$.
Then

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b p(x)\mathrm{d}x$$

We can also use a general interpolating polynomial of degree $n-1$ given in Lagrange form

$$p(x) = \sum_{k=1}^{n} L_k(x) y_k$$

with Lagrange basis polynomials $L_k(x) = \prod_{j=1, j\neq k}^{n} \frac{x-x_j}{x_k-x_j}$, $k = 1, \ldots, n$. Then

$$\int_a^b f(x)\mathrm{d}x \;\approx\; \int_a^b p(x)\mathrm{d}x = \int_a^b \sum_{k=1}^{n} L_k(x) f(x_k)\mathrm{d}x$$

We can also use a general interpolating polynomial of degree $n-1$ given in Lagrange form

$$p(x) = \sum_{k=1}^{n} L_k(x) y_k$$

with Lagrange basis polynomials $L_k(x) = \prod_{j=1, j\neq k}^{n} \frac{x-x_j}{x_k-x_j}$, $k = 1, \ldots, n$. Then

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &\approx \int_a^b p(x)\mathrm{d}x = \int_a^b \sum_{k=1}^{n} L_k(x) f(x_k)\mathrm{d}x \\ &= \sum_{k=1}^{n} f(x_k) \int_a^b L_k(x)\mathrm{d}x\end{aligned}$$

We can also use a general interpolating polynomial of degree $n-1$ given in Lagrange form

$$p(x) = \sum_{k=1}^{n} L_k(x) y_k$$

with Lagrange basis polynomials $L_k(x) = \prod_{j=1, j\neq k}^{n} \frac{x - x_j}{x_k - x_j}$, $k = 1, \ldots, n$. Then

$$\begin{aligned} \int_a^b f(x)\mathrm{d}x &\approx \int_a^b p(x)\mathrm{d}x = \int_a^b \sum_{k=1}^{n} L_k(x) f(x_k)\mathrm{d}x \\ &= \sum_{k=1}^{n} f(x_k) \int_a^b L_k(x)\mathrm{d}x = \sum_{k=1}^{n} f(x_k) w_k, \end{aligned}$$

We can also use a general interpolating polynomial of degree $n-1$ given in Lagrange form

$$p(x) = \sum_{k=1}^{n} L_k(x) y_k$$

with Lagrange basis polynomials $L_k(x) = \prod_{j=1, j\neq k}^{n} \frac{x-x_j}{x_k-x_j}$, $k = 1, \ldots, n$. Then

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &\approx \int_a^b p(x)\mathrm{d}x = \int_a^b \sum_{k=1}^{n} L_k(x) f(x_k)\mathrm{d}x \\ &= \sum_{k=1}^{n} f(x_k) \int_a^b L_k(x)\mathrm{d}x = \sum_{k=1}^{n} f(x_k) w_k,\end{aligned}$$

where the integration weights $w_k = \int_a^b L_k(x)\mathrm{d}x$ can be computed since the integrands $L_k(x)$ are simple polynomials.

We can also use a general interpolating polynomial of degree $n-1$ given in Lagrange form

$$p(x) = \sum_{k=1}^{n} L_k(x) y_k$$

with Lagrange basis polynomials $L_k(x) = \prod_{j=1, j\neq k}^{n} \frac{x-x_j}{x_k-x_j}$, $k = 1, \ldots, n$.
Then

$$\begin{aligned} \int_a^b f(x)\mathrm{d}x &\approx \int_a^b p(x)\mathrm{d}x = \int_a^b \sum_{k=1}^{n} L_k(x) f(x_k)\mathrm{d}x \\ &= \sum_{k=1}^{n} f(x_k) \int_a^b L_k(x)\mathrm{d}x = \sum_{k=1}^{n} f(x_k) w_k, \end{aligned}$$

where the integration weights $w_k = \int_a^b L_k(x)\mathrm{d}x$ can be computed since the integrands $L_k(x)$ are simple polynomials.
If the interpolation nodes x_k are equally spaced, the resulting formulas are known as Newton-Cotes formulas.

Since we used polynomial interpolants on subintervals (or piecewise polynomial interpolants on $[a, b]$) for all of our integration methods, they have the following properties by construction:

- The left and right endpoint methods and the midpoint method are exact if the integrand is a (piecewise) constant function.
- The composite trapezoidal rule is exact if the integrand is a (piecewise) linear function.
- The composite Simpson's rule is exact if the integrand is a (piecewise) quadratic function.

Remark

It turns out that the midpoint method is even exact for piecewise linear functions, and Simpson's rule is exact for cubic polynomials!

Outline

A surprisingly simple way to improve the accuracy of many numerical methods (not just integration!) is provided by the general idea of Richardson extrapolation.

A surprisingly simple way to improve the accuracy of many numerical methods (not just integration!) is provided by the general idea of Richardson extrapolation.

Here one runs the same approximation method twice to get two numerical approximations:

- F_n (or F_h), based on a subdivision into n subintervals (or intervals of length h), and

A surprisingly simple way to improve the accuracy of many numerical methods (not just integration!) is provided by the general idea of Richardson extrapolation.

Here one runs the same approximation method twice to get two numerical approximations:

- F_n (or F_h), based on a subdivision into n subintervals (or intervals of length h), and
- F_{2n} (or $F_{\frac{h}{2}}$), based on a subdivision into $2n$ subintervals (or intervals of length $\frac{h}{2}$).

A surprisingly simple way to improve the accuracy of many numerical methods (not just integration!) is provided by the general idea of Richardson extrapolation.

Here one runs the same approximation method twice to get two numerical approximations:

- F_n (or F_h), based on a subdivision into n subintervals (or intervals of length h), and
- F_{2n} (or $F_{\frac{h}{2}}$), based on a subdivision into $2n$ subintervals (or intervals of length $\frac{h}{2}$).

We will first discuss the general idea, and then look at examples for the left and right endpoint, midpoint, and trapezoidal rules.

A surprisingly simple way to improve the accuracy of many numerical methods (not just integration!) is provided by the general idea of Richardson extrapolation.

Here one runs the same approximation method twice to get two numerical approximations:

- F_n (or F_h), based on a subdivision into n subintervals (or intervals of length h), and
- F_{2n} (or $F_{\frac{h}{2}}$), based on a subdivision into $2n$ subintervals (or intervals of length $\frac{h}{2}$).

We will first discuss the general idea, and then look at examples for the left and right endpoint, midpoint, and trapezoidal rules.

In the next section we discuss the MATLAB function `quad` which is based on Richardson extrapolation for Simpson's rule.

Richardson extrapolation: the general principle

Assume we have some numerical method whose output F_h approximates an unknown quantity F according to

$$F = F_h + \underbrace{\mathcal{O}(h^p)}_{=E_h} \tag{3}$$

for some power $p \geq 1$, i.e., we have a truncation error E_h of order $\mathcal{O}(h^p)$.

Richardson extrapolation: the general principle

Assume we have some numerical method whose output F_h approximates an unknown quantity F according to

$$F = F_h + \underbrace{\mathcal{O}(h^p)}_{=E_h} \qquad (3)$$

for some power $p \geq 1$, i.e., we have a truncation error E_h of order $\mathcal{O}(h^p)$.

Consider two approximate values F_h and $F_{\frac{h}{2}}$.

Richardson extrapolation: the general principle

Assume we have some numerical method whose output F_h approximates an unknown quantity F according to

$$F = F_h + \underbrace{\mathcal{O}(h^p)}_{=E_h} \tag{3}$$

for some power $p \geq 1$, i.e., we have a truncation error E_h of order $\mathcal{O}(h^p)$.

Consider two approximate values F_h and $F_{\frac{h}{2}}$.

Then the truncation error at the $\frac{h}{2}$ level satisfies

$$E_{\frac{h}{2}} \approx c \left(\frac{h}{2}\right)^p$$

Richardson extrapolation: the general principle

Assume we have some numerical method whose output F_h approximates an unknown quantity F according to

$$F = F_h + \underbrace{\mathcal{O}(h^p)}_{=E_h} \tag{3}$$

for some power $p \geq 1$, i.e., we have a truncation error E_h of order $\mathcal{O}(h^p)$.

Consider two approximate values F_h and $F_{\frac{h}{2}}$.

Then the truncation error at the $\frac{h}{2}$ level satisfies

$$E_{\frac{h}{2}} \approx c\left(\frac{h}{2}\right)^p = c\frac{h^p}{2^p}$$

Richardson extrapolation: the general principle

Assume we have some numerical method whose output F_h approximates an unknown quantity F according to

$$F = F_h + \underbrace{\mathcal{O}(h^p)}_{=E_h} \tag{3}$$

for some power $p \geq 1$, i.e., we have a truncation error E_h of order $\mathcal{O}(h^p)$.

Consider two approximate values F_h and $F_{\frac{h}{2}}$.

Then the truncation error at the $\frac{h}{2}$ level satisfies

$$E_{\frac{h}{2}} \approx c\left(\frac{h}{2}\right)^p = c\frac{h^p}{2^p} \approx \frac{1}{2^p}E_h. \tag{4}$$

Therefore, using (3) and the error relation (4) we have

$$F - F_{\frac{h}{2}} \overset{(3) \text{ for } \frac{h}{2}}{=} E_{\frac{h}{2}}$$

Therefore, using (3) and the error relation (4) we have

$$F - F_{\frac{h}{2}} \overset{(3)\text{ for }\frac{h}{2}}{=} E_{\frac{h}{2}} \overset{(4)}{\approx} \frac{1}{2^p} E_h$$

Therefore, using (3) and the error relation (4) we have

$$F - F_{\frac{h}{2}} \overset{(3) \text{ for } \frac{h}{2}}{=} E_{\frac{h}{2}} \overset{(4)}{\approx} \frac{1}{2^p} E_h \overset{(3) \text{ for } h}{=} \frac{1}{2^p}(F - F_h).$$

Therefore, using (3) and the error relation (4) we have

$$F - F_{\frac{h}{2}} \overset{(3) \text{ for } \frac{h}{2}}{=} E_{\frac{h}{2}} \overset{(4)}{\approx} \frac{1}{2^p} E_h \overset{(3) \text{ for } h}{=} \frac{1}{2^p}(F - F_h).$$

This implies

$$F\left(1 - \frac{1}{2^p}\right) \approx F_{\frac{h}{2}} - \frac{1}{2^p} F_h$$

Therefore, using (3) and the error relation (4) we have

$$F - F_{\frac{h}{2}} \overset{(3)\text{ for }\frac{h}{2}}{=} E_{\frac{h}{2}} \overset{(4)}{\approx} \frac{1}{2^p} E_h \overset{(3)\text{ for }h}{=} \frac{1}{2^p}(F - F_h).$$

This implies

$$F\left(1 - \frac{1}{2^p}\right) \approx F_{\frac{h}{2}} - \frac{1}{2^p} F_h$$

or

$$F \approx \frac{2^p}{2^p - 1}\left[F_{\frac{h}{2}} - \frac{F_h}{2^p}\right].$$

Therefore, using (3) and the error relation (4) we have

$$F - F_{\frac{h}{2}} \overset{(3) \text{ for } \frac{h}{2}}{=} E_{\frac{h}{2}} \overset{(4)}{\approx} \frac{1}{2^p} E_h \overset{(3) \text{ for } h}{=} \frac{1}{2^p}(F - F_h).$$

This implies

$$F\left(1 - \frac{1}{2^p}\right) \approx F_{\frac{h}{2}} - \frac{1}{2^p} F_h$$

or

$$F \approx \frac{2^p}{2^p - 1}\left[F_{\frac{h}{2}} - \frac{F_h}{2^p}\right].$$

The latter can be rewritten as

$$F \approx F_{\frac{h}{2}} + \frac{1}{2^p - 1}\left[F_{\frac{h}{2}} - F_h\right]. \tag{5}$$

This is the Richardson extrapolation formula, a weighted average of $F_{\frac{h}{2}}$ and F_h.

Therefore, using (3) and the error relation (4) we have

$$F - F_{\frac{h}{2}} \overset{(3) \text{ for } \frac{h}{2}}{=} E_{\frac{h}{2}} \overset{(4)}{\approx} \frac{1}{2^p} E_h \overset{(3) \text{ for } h}{=} \frac{1}{2^p}(F - F_h).$$

This implies

$$F\left(1 - \frac{1}{2^p}\right) \approx F_{\frac{h}{2}} - \frac{1}{2^p} F_h$$

or

$$F \approx \frac{2^p}{2^p - 1}\left[F_{\frac{h}{2}} - \frac{F_h}{2^p}\right].$$

The latter can be rewritten as

$$F \approx F_{\frac{h}{2}} + \frac{1}{2^p - 1}\left[F_{\frac{h}{2}} - F_h\right]. \tag{5}$$

This is the Richardson extrapolation formula, a weighted average of $F_{\frac{h}{2}}$ and F_h.

It is considerably more accurate than either $F_{\frac{h}{2}}$ or F_h. In fact, it is constructed to yield at least $\mathcal{O}(h^{p+1})$ accuracy.

Remark

From the Richardson extrapolation formula (5) we see that

$$E_{\frac{h}{2}} \approx \frac{1}{2^p - 1}\left[F_{\frac{h}{2}} - F_h\right].$$

Remark

From the Richardson extrapolation formula (5) we see that

$$E_{\frac{h}{2}} \approx \frac{1}{2^p - 1}\left[F_{\frac{h}{2}} - F_h\right].$$

Therefore, we can use (4) to provide an estimate for the truncation error E_h, namely

$$E_h \approx 2^p E_{\frac{h}{2}}$$

Remark

From the Richardson extrapolation formula (5) we see that

$$E_{\frac{h}{2}} \approx \frac{1}{2^p - 1}\left[F_{\frac{h}{2}} - F_h\right].$$

Therefore, we can use (4) to provide an estimate for the truncation error E_h, namely

$$E_h \approx 2^p E_{\frac{h}{2}} \approx \frac{2^p}{2^p - 1}\left[F_{\frac{h}{2}} - F_h\right].$$

Remark

From the Richardson extrapolation formula (5) we see that

$$E_{\frac{h}{2}} \approx \frac{1}{2^p - 1}\left[F_{\frac{h}{2}} - F_h\right].$$

Therefore, we can use (4) to provide an estimate for the truncation error E_h, namely

$$E_h \approx 2^p E_{\frac{h}{2}} \approx \frac{2^p}{2^p - 1}\left[F_{\frac{h}{2}} - F_h\right].$$

Note that it is better to be conservative and use this latter error estimate at the h level.

Remark

From the Richardson extrapolation formula (5) we see that

$$E_{\frac{h}{2}} \approx \frac{1}{2^p - 1} \left[F_{\frac{h}{2}} - F_h \right].$$

Therefore, we can use (4) to provide an estimate for the truncation error E_h, namely

$$E_h \approx 2^p E_{\frac{h}{2}} \approx \frac{2^p}{2^p - 1} \left[F_{\frac{h}{2}} - F_h \right].$$

Note that it is better to be conservative and use this latter error estimate at the h level.
We will use this idea to obtain an adaptive integration algorithm in the next section.

Example

We know that the left endpoint method

$$L_n(f) = \sum_{i=1}^{n} hf(x_{i-1})$$

is order $\mathcal{O}(h)$ accurate so that $p = 1$.

Example

We know that the left endpoint method

$$L_n(f) = \sum_{i=1}^{n} hf(x_{i-1})$$

is order $\mathcal{O}(h)$ accurate so that $p = 1$.
According to our discussion above we can use the Richardson extrapolant

$$L \approx 2L_{2n} - L_n$$

to improve the accuracy of the left endpoint method.

Example

We know that the left endpoint method

$$L_n(f) = \sum_{i=1}^{n} hf(x_{i-1})$$

is order $\mathcal{O}(h)$ accurate so that $p = 1$.
According to our discussion above we can use the Richardson extrapolant

$$L \approx 2L_{2n} - L_n$$

to improve the accuracy of the left endpoint method.
The truncation error of L_n can be estimated via

$$E_n \approx 2\left[L_{2n} - L_n\right].$$

Example

We know that the left endpoint method

$$L_n(f) = \sum_{i=1}^{n} hf(x_{i-1})$$

is order $\mathcal{O}(h)$ accurate so that $p = 1$.
According to our discussion above we can use the Richardson extrapolant

$$L \approx 2L_{2n} - L_n$$

to improve the accuracy of the left endpoint method.
The truncation error of L_n can be estimated via

$$E_n \approx 2\left[L_{2n} - L_n\right].$$

This and similar formulas for the other integration methods are illustrated in the Maple worksheet `Integration.mw`.

Outline

The main ingredient for MATLAB's basic numerical integration function `quad` is Richardson extrapolation of the basic Simpson rule.

The main ingredient for MATLAB's basic numerical integration function `quad` is Richardson extrapolation of the basic Simpson rule.
If S_2 is Simpson's rule applied to $x_0 = a$, $x_1 = \frac{a+b}{2}$ and $x_2 = b$, i.e.,

$$S_2 = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right],$$

with[1] $h = \frac{b-a}{2}$,

[1] Note that in [NCM] $h = b - a$, so formulas differ slightly

The main ingredient for MATLAB's basic numerical integration function `quad` is Richardson extrapolation of the basic Simpson rule.
If S_2 is Simpson's rule applied to $x_0 = a$, $x_1 = \frac{a+b}{2}$ and $x_2 = b$, i.e.,

$$S_2 = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right],$$

with[1] $h = \frac{b-a}{2}$, and S_4 is a refined (composite) Simpson's rule for the five points $\tilde{x}_i = a + i\frac{h}{2}$, $i = 0, 1, \ldots, 4$, i.e.,

$$S_4 = \frac{h}{6}\left[f(\tilde{x}_0) + 4f(\tilde{x}_1) + 2f(\tilde{x}_2) + 4f(\tilde{x}_3) + f(\tilde{x}_4)\right],$$

[1] Note that in [NCM] $h = b - a$, so formulas differ slightly

The main ingredient for MATLAB's basic numerical integration function `quad` is Richardson extrapolation of the basic Simpson rule.
If S_2 is Simpson's rule applied to $x_0 = a$, $x_1 = \frac{a+b}{2}$ and $x_2 = b$, i.e.,

$$S_2 = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right],$$

with[1] $h = \frac{b-a}{2}$, and S_4 is a refined (composite) Simpson's rule for the five points $\tilde{x}_i = a + i\frac{h}{2}$, $i = 0, 1, \ldots, 4$, i.e.,

$$S_4 = \frac{h}{6}\left[f(\tilde{x}_0) + 4f(\tilde{x}_1) + 2f(\tilde{x}_2) + 4f(\tilde{x}_3) + f(\tilde{x}_4)\right],$$

then Richardson's formula (5) with $p = 4$ yields

$$S = S_4 + \frac{1}{15}(S_4 - S_2).$$

[1] Note that in [NCM] $h = b - a$, so formulas differ slightly

The main ingredient for MATLAB's basic numerical integration function `quad` is Richardson extrapolation of the basic Simpson rule.
If S_2 is Simpson's rule applied to $x_0 = a$, $x_1 = \frac{a+b}{2}$ and $x_2 = b$, i.e.,

$$S_2 = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right],$$

with[1] $h = \frac{b-a}{2}$, and S_4 is a refined (composite) Simpson's rule for the five points $\tilde{x}_i = a + i\frac{h}{2}$, $i = 0, 1, \ldots, 4$, i.e.,

$$S_4 = \frac{h}{6}\left[f(\tilde{x}_0) + 4f(\tilde{x}_1) + 2f(\tilde{x}_2) + 4f(\tilde{x}_3) + f(\tilde{x}_4)\right],$$

then Richardson's formula (5) with $p = 4$ yields

$$S = S_4 + \frac{1}{15}(S_4 - S_2).$$

This is in fact a sixth-order, $\mathcal{O}(h^6)$, Newton-Cotes formula (Weddle's rule).

[1] Note that in [NCM] $h = b - a$, so formulas differ slightly

The main part of `quadtx.m` is the recursively called function `quadtxstep` that performs the extrapolated Simpson rule and refines the intervals when needed:

```
h = b - a;
c = (a + b)/2;
fd = F((a+c)/2,varargin{:});
fe = F((c+b)/2,varargin{:});
Q1 = h/6 * (fa + 4*fc + fb);                        % Simpson S_2
Q2 = h/12 * (fa + 4*fd + 2*fc + 4*fe + fb);  % Simpson S_4
if abs(Q2 - Q1) <= tol        % error estimate small enough
   Q  = Q2 + (Q2 - Q1)/15;    % extrapolate
else                          % subdivide at interval midpoint c
   [Qa,ka] = quadtxstep(F, a, c, tol, fa, fd, fc, varargin{:});
   [Qb,kb] = quadtxstep(F, c, b, tol, fc, fe, fb, varargin{:});
   Q  = Qa + Qb;
end
```

The main part of `quadtx.m` is the recursively called function `quadtxstep` that performs the extrapolated Simpson rule and refines the intervals when needed:

```
h = b - a;
c = (a + b)/2;
fd = F((a+c)/2,varargin{:});
fe = F((c+b)/2,varargin{:});
Q1 = h/6 * (fa + 4*fc + fb);                        % Simpson S_2
Q2 = h/12 * (fa + 4*fd + 2*fc + 4*fe + fb);   % Simpson S_4
if abs(Q2 - Q1) <= tol        % error estimate small enough
   Q  = Q2 + (Q2 - Q1)/15;    % extrapolate
else                          % subdivide at interval midpoint c
   [Qa,ka] = quadtxstep(F, a, c, tol, fa, fd, fc, varargin{:});
   [Qb,kb] = quadtxstep(F, c, b, tol, fc, fe, fb, varargin{:});
   Q  = Qa + Qb;
end
```

We illustrate the use of `quadtx` in the MATLAB file `QuadDemo.m`.

The main part of `quadtx.m` is the recursively called function `quadtxstep` that performs the extrapolated Simpson rule and refines the intervals when needed:

```
h = b - a;
c = (a + b)/2;
fd = F((a+c)/2,varargin{:});
fe = F((c+b)/2,varargin{:});
Q1 = h/6 * (fa + 4*fc + fb);                          % Simpson S_2
Q2 = h/12 * (fa + 4*fd + 2*fc + 4*fe + fb);   % Simpson S_4
if abs(Q2 - Q1) <= tol          % error estimate small enough
   Q  = Q2 + (Q2 - Q1)/15;      % extrapolate
else                            % subdivide at interval midpoint c
   [Qa,ka] = quadtxstep(F, a, c, tol, fa, fd, fc, varargin{:});
   [Qb,kb] = quadtxstep(F, c, b, tol, fc, fe, fb, varargin{:});
   Q  = Qa + Qb;
end
```

We illustrate the use of `quadtx` in the MATLAB file `QuadDemo.m`.
The [NCM] program `quadgui` illustrates graphically how the interval is refined adaptively.

The latest addition to MATLAB's quadrature methods is the function `quadgk` developed by Larry Shampine.
This method has a number of advantages over `quad`:

- It is vectorized to do the function evaluations for all subinterval simultaneously.
- It starts with a higher resolution, and is therefore less affected by "difficult" integrands.
- It takes relative error tolerances.
- It can handle infinite intervals and endpoint singularities.
- Most of all, `quadgk` is much faster and more reliable than `quad` or `quadl`.

`quadgk` is described in the recent paper [Shampine] (where the method is referred to as `quadva`).

Outline

If the integrand is given only in the form of discrete data, then our fancy adaptive methods will not work (since we can't choose where we evaluate the integrand — we have to work with the given values).

If the integrand is given only in the form of discrete data, then our fancy adaptive methods will not work (since we can't choose where we evaluate the integrand — we have to work with the given values).

The simplest solution is given by a composite trapezoidal rule (integral of piecewise linear interpolant):

$$T = \sum_{i=1}^{n-1} h_i \frac{y_i + y_{i+1}}{2},$$

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

If the integrand is given only in the form of discrete data, then our fancy adaptive methods will not work (since we can't choose where we evaluate the integrand — we have to work with the given values).
The simplest solution is given by a composite trapezoidal rule (integral of piecewise linear interpolant):

$$T = \sum_{i=1}^{n-1} h_i \frac{y_i + y_{i+1}}{2},$$

where $h_i = x_{i+1} - x_i$. The entire MATLAB code for this method is (assuming the data vectors `x` and `y` are given)

```
T = sum(diff(x).*(y(1:end-1)+y(2:end))/2)
```

Note how vectorization makes the code very compact. This code is the basis for the MATLAB command `trapz`.

If the integrand is given only in the form of discrete data, then our fancy adaptive methods will not work (since we can't choose where we evaluate the integrand — we have to work with the given values).
The simplest solution is given by a composite trapezoidal rule (integral of piecewise linear interpolant):

$$T = \sum_{i=1}^{n-1} h_i \frac{y_i + y_{i+1}}{2},$$

where $h_i = x_{i+1} - x_i$. The entire MATLAB code for this method is (assuming the data vectors `x` and `y` are given)

```
T = sum(diff(x).*(y(1:end-1)+y(2:end))/2)
```

Note how vectorization makes the code very compact. This code is the basis for the MATLAB command `trapz`.
One can also implement integration of other interpolants (see [NCM] for a discussion of `pchip` and `spline` interpolants).

References I

C. Moler.
Numerical Computing with MATLAB.
SIAM, Philadelphia, 2004.
Also http://www.mathworks.com/moler/.

L. F. Shampine.
Vectorized adaptive quadrature in MATLAB.
J. Comput. Appl. Math. **211** (2008), 131–140.