To evaluate the test integral

$$\int_0^1 x^3 e^{-x} dx = 0.1139289\ldots,$$

the sample program 10.1 calls function `qTrapz`, which implements the trapezoidal rule.

Along with the integration limits, `a` and `b`, and the number of integration points, `n`, the function `qTrapz` receives the actual name of the user function that codes the integrand by the procedural argument `Func`. In generating the integration nodes, instead of the recursive incrementation $x_{i+1} = x_i + h$, the nonrecursive expression $a + ih$ is preferable, since it reduces roundoff errors (while the shifted index spares a subtraction) and can be directly passed as an argument in calls to function `Func`.

10.3 The Newton–Cotes Quadrature Formulas

The trapezoidal rule can be shown to be just the lowest-order member of an entire family, called *Newton–Cotes quadrature formulas*, devised for integrands specified on regular integration meshes. The general underlying idea is to replace the integrand by a suitable interpolating polynomial based on the equidistant integrand values.

With a view to evaluating the definite integral

$$I = \int_a^b f(x)dx, \tag{10.4}$$

we start by dividing the interval $[a, b]$ into $(n - 1)$ equal subintervals of length

$$h = (b - a)/(n - 1), \tag{10.5}$$

by the equally spaced abscissas

$$x_i = a + (i - 1)h, \quad i = 1, 2, \ldots, n. \tag{10.6}$$

Assuming that the integrand values $f_i \equiv f(x_i)$ at the integration nodes x_i are known, we approximate the integrand by the *Lagrange interpolating polynomial* (see Section 9.2):

$$P_{n-1}(x) = \sum_{i=1}^{n} \frac{\prod_{j\neq i}^{n}(x - x_j)}{\prod_{j\neq i}^{n}(x_i - x_j)} f_i.$$

By introducing the dimensionless variable

$$q = (x - a)/h, \quad q \in [0, n - 1],$$

the function arguments may be expressed as $x = a + qh$ and the products occurring in the coefficients of the Lagrange polynomial can be rewritten as

$$\prod_{j\neq i}^{n}(x - x_j) = h^{n-1}\prod_{j\neq i}^{n}[q - (j-1)],$$

$$\prod_{j\neq i}^{n}(x_i - x_j) = h^{n-1}\prod_{j\neq i}^{n}(i-j) = (-1)^{n-i}h^{n-1}\prod_{j=1}^{i-1}(i-j)\prod_{j=i+1}^{n}(j-i)$$

$$= (-1)^{n-i}h^{n-1}(i-1)!(n-i)!$$

The Lagrange polynomial thus takes the form

$$P_{n-1}(x) = \sum_{i=1}^{n}\frac{\prod_{j\neq i}^{n}[q-(j-1)]}{(-1)^{n-i}(i-1)!(n-i)!}f_i, \tag{10.7}$$

and the following approximation to the integral is obtained:

$$\int_a^b f(x)dx \approx \int_a^b P_{n-1}(x)dx = \sum_{i=1}^{n}A_i f_i. \tag{10.8}$$

The coefficients A_i weighting the integrand values can be successively transformed,

$$A_i = \int_a^b \frac{\prod_{j\neq i}^{n}[q-(j-1)]dx}{(-1)^{n-i}(i-1)!(n-i)!} = \frac{h\int_0^{n-1}\prod_{j\neq i}^{n}[q-(j-1)]dq}{(-1)^{n-i}(i-1)!(n-i)!}, \tag{10.9}$$

and, by factoring out explicitly the extent of the integration interval, $A_i = (b-a)H_i$, the classical *Newton–Cotes quadrature formula* results:

$$\int_a^b f(x)dx \approx (b-a)\sum_{i=1}^{n}H_i f_i. \tag{10.10}$$

The corresponding *Cotes coefficients* are given by

$$H_i = \frac{\int_0^{n-1}\prod_{j\neq i}^{n}[q-(j-1)]dq}{(-1)^{n-i}(i-1)!(n-i)!(n-1)}, \quad i = 1, 2, \ldots, n. \tag{10.11}$$

It should be noted that, as per definition, the Cotes coefficients do neither depend on the integrated function, nor on the integration interval. In addition, they satisfy:

$$\sum_{i=1}^{n}H_i = 1, \quad H_i = H_{n-i+1}. \tag{10.12}$$

The first property, reflecting the *normalization* of the coefficients, may be demonstrated by simply considering the particular integrand $f(x) \equiv 1$ in the general Newton–Cotes formula 10.10, while the second property, implying the *symmetry* of the coefficient sequence with respect to its median, follows from the

very definition of the coefficients, Equation 10.11, which remains invariant when replacing index i with its symmetric $n - i + 1$.

10.4 Trapezoidal Rule

Next, we consider the Newton–Cotes quadrature (10.10) for the particular case $n = 2$, which implies that the only integrand values used are the ones at the integration interval limits. The two involved Cotes coefficients (10.11) take equal values:

$$H_1 = -\int_0^1 (q-1)dq = \frac{1}{2}, \tag{10.13}$$

$$H_2 = \int_0^1 q dq = \frac{1}{2}, \tag{10.14}$$

and therewith, the *trapezoidal formula* results:

$$\int_{x_1}^{x_2} f(x)dx \approx \frac{h}{2}(f_1 + f_2). \tag{10.15}$$

The name of the method comes from the fact that, as seen from Figure 10.2, the formula can be obtained directly by replacing the function $f(x)$ with the segment connecting the points (x_1, f_1) and (x_2, f_2), and the integral thus amounts to the area of the trapeze formed by the segment and its projection onto the x-axis.

The error associated with the trapezoidal formula 10.15, depicted as hatched in the plot of Figure 10.2, may be established imposing on $f(x)$ and its first two derivatives to be continuous on $[a, b]$ ($f(x) \in C^{(2)}[a, b]$). Aiming to establish a practical measure for the precision of a quadrature formula, one typically expresses the error term as a function of the integration step size h (Demidovich and Maron, 1987):

$$R(h) = \int_{x_1}^{x_1+h} f(x)dx - \frac{h}{2}[f(x_1) + f(x_1 + h)].$$

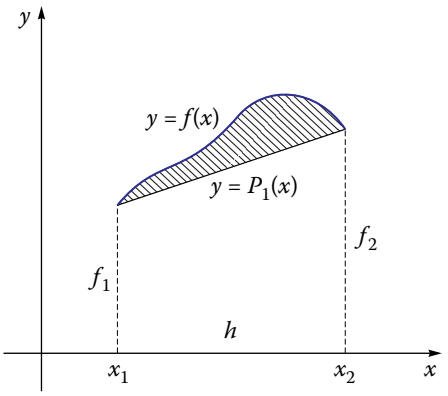

FIGURE 10.2 The trapezoidal formula approximates the integrand by the segment connecting the points (x_1, f_1) and (x_2, f_2).

By taking twice the derivative of the above expression with respect to h, we get:

$$R'(h) = \frac{1}{2}[f(x_1 + h) - f(x_1)] - \frac{h}{2}f'(x_1 + h),$$
$$R''(h) = -\frac{h}{2}f''(x_1 + h).$$

We now integrate $R''(h)$ twice, noting that $R(0) = R'(0) = 0$. Employing the *mean value theorem* (which essentially states that there is at least one point on an arc of a differentiable curve at which the derivative is equal to the average derivative of the arc), it results successively:

$$R'(h) = R'(0) + \int_0^h R''(u)du = -\frac{1}{2}f''(\xi_1)\int_0^h udu = -\frac{1}{4}h^2 f''(\xi_1),$$
$$R(h) = R(0) + \int_0^h R'(u)du = -\frac{1}{4}f''(\xi)\int_0^h u^2 du = -\frac{1}{12}h^3 f''(\xi),$$

where $\xi, \xi_1 \in (x_1, x_1 + h)$. Finally, we obtain for the error term of the trapezoidal formula:

$$R(h) = -\frac{1}{12}h^3 f''(\xi), \quad \xi \in (x_1, x_2). \tag{10.16}$$

It appears that the error term has opposite sign to the second derivative in the interval (x_1, x_2) and, consequently, the trapezoidal formula overestimates the integral if $f'' > 0$ and underestimates it otherwise.

The trapezoidal formula 10.15 is, obviously, of no practical interest as such, having rather a theoretical importance. Instead, by partitioning the integration interval $[a, b]$ by n equidistant points x_i and making use of the *additive property of integrals for subintervals*, one regains the *trapezoidal rule* (derived heuristically in Section 10.2),

$$\int_a^b f(x)dx \approx h\left[\frac{f_1}{2} + \sum_{i=2}^{n-1} f_i + \frac{f_n}{2}\right], \tag{10.17}$$

where h is the spacing of the integration points.

The error term associated with the trapezoidal rule cumulates the errors corresponding to the $(n-1)$ integration subintervals, namely,

$$R_T(h) = -\frac{(n-1)}{12}h^3 f''(\xi) = -\frac{(b-a)}{12}h^2 f''(\xi), \quad \xi \in [a, b], \tag{10.18}$$

and shows the characteristic $O(h^2)$ order. Even though such an error term cannot be used as a practical estimate of the integration error, it implies that, for instance, by halving the mesh spacing h, the error roughly reduces by a factor of 4. Moreover, the error term enables by the *step-halving technique* the adaptive control of the integration mesh itself. This consists of calculating the integral for a given h (corresponding to n nodes) and comparing it with the result for $h/2$ (corresponding to $(2n-1)$ nodes). The iterative halving of the mesh spacing is to be continued, in principle, until the relative difference between two consecutive approximations, regarded as on error estimate, drops under an admissible value ε.

10.5 Simpson's Rule

Considering the Newton–Cotes formulas 10.10 and 10.11 in particular for $n = 3$ mesh points, one obtains a quadrature method superior in precision to the trapezoidal formula. The corresponding Cotes coefficients are

$$H_1 = \frac{1}{4}\int_0^2 (q-1)(q-2)dq = \frac{1}{6}, \tag{10.19}$$

$$H_2 = -\frac{1}{2}\int_0^2 q(q-2)dq = \frac{2}{3}, \tag{10.20}$$

$$H_3 = \frac{1}{4}\int_0^2 q(q-1)dq = \frac{1}{6}, \tag{10.21}$$

and, given that $b - a \equiv x_3 - x_1 = 2h$, *Simpson's formula* results:

$$\int_{x_1}^{x_3} f(x)dx \approx \frac{h}{3}(f_1 + 4f_2 + f_3). \tag{10.22}$$

From a geometrical viewpoint, Simpson's formula implies replacing the curve $y = f(x)$ by the parabola $y = P_2(x)$, with the Lagrange polynomial $P_2(x)$ defined by the three points (x_1, f_1), (x_2, f_2), and (x_3, f_3).

Assuming the integrand $f(x)$ to be continuous together with its first four derivatives on $[a, b]$ ($f(x) \in C^{(4)}[a, b]$), one obtains an estimate of the error term for Simpson's formula by a technique similar to the one employed in the case of the trapezoidal formula (see Demidovich and Maron 1987):

$$R = -\frac{1}{90}h^5 f^{(4)}(\xi), \quad \xi \in [a, b]. \tag{10.23}$$

While the error depends on h^3 for the trapezoidal formula, it scales with h^5 for Simpson's formula and the latter can be shown to be exact not only for second-order polynomials, but also for third-order polynomials.

To establish a method of practical interest, usable with controllable precision for arbitrary integrands, one generalizes Simpson's formula 10.22 in a similar manner to the trapezoidal rule by making use of the *additive property of integrals for subintervals*. Concretely, dividing the interval $[a, b]$ by an *odd* number, $n = 2m + 1$, of equally spaced points,

$$x_i = a + (i-1)h, \quad i = 1, 2, \ldots, n,$$

separated by

$$h = \frac{b-a}{n-1} = \frac{b-a}{2m},$$

one can apply Simpson's formula 10.22 to each of the m double subintervals (of length $2h$) determined by three consecutive mesh points: $[x_1, x_3], \ldots, [x_{n-2}, x_n]$. Geometrically, this approach boils down to approximating the integrand by a piecewise parabolic function and, obviously, the generalization can be

solely based on a mesh with *odd* number of points. Thus, the integral over the entire interval $[a, b]$ may be composed as

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f_1 + 4f_2 + f_3\right) + \frac{h}{3}\left(f_3 + 4f_4 + f_5\right) + \cdots$$
$$+\frac{h}{3}\left(f_{n-4} + 4f_{n-3} + f_{n-2}\right) + \frac{h}{3}\left(f_{n-2} + 4f_{n-1} + f_n\right).$$

By regrouping the terms, one obtains *Simpson's rule*:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f_1 + 4\sigma_2 + 2\sigma_1 + f_n\right), \tag{10.24}$$

where

$$\sigma_1 = \sum_{i=3,5}^{n-2} f_i, \quad \sigma_2 = \sum_{i=2,4}^{n-1} f_i. \tag{10.25}$$

Sum σ_1 is composed only of the odd-index terms, while σ_2 cumulates the even-index ones and contains one term more than σ_1.

Assuming $f(x)$ to be continuous of class $C^{(4)}[a, b]$, the error term of Simpson's rule results by summing errors (10.23) for the m double subintervals of length $2h$:

$$R_S = -\frac{m}{90}h^5 f^{(4)}(\xi) = -\frac{(b-a)}{180}h^4 f^{(4)}(\xi), \quad \xi \in [a, b]. \tag{10.26}$$

Since it implies derivatives at unspecified interior points, the remainder is not an operational error estimate, but should be merely regarded as an indication of the *order of the global error*, that is, $O(h^4)$. Correspondingly, the *degree of precision* of Simpson's rule is three, implying that the composite formula is *exact* for polynomials up to degree three.

With a view to an efficient implementation, the indexes in the sums σ_1 and σ_2 are shifted by -1, so as to eliminate the subtraction $(i - 1)$ in the definition of the integration points:

$$\sigma_1 = \sum_{i=2,4}^{n-3} f\left(a + ih\right), \quad \sigma_2 = \sum_{i=1,3}^{n-2} f\left(a + ih\right). \tag{10.27}$$

Listing 10.2 Function Integrator Based on Simpson's Rule (Python Coding)

```
#==============================================================================
def qSimpson(Func, a, b, n):
#------------------------------------------------------------------------------
#  Integrates function Func on interval [a,b] using Simpson's rule with n
#  (odd) integration points
#------------------------------------------------------------------------------
   if (n % 2 == 0): n += 1                                # increment n if even

   h = (b-a)/(n-1)
   s1 = s2 = 0e0
   for i in range(2,n-2,2): s1 += Func(a + i*h)              # odd-index sum
   for i in range(1,n-1,2): s2 += Func(a + i*h)             # even-index sum

   return (h/3)*(Func(a) + 4*s2 + 2*s1 + Func(b))
```