Chapter 6. Numerical Integration

6.1. Introduction

Many undergraduates prefer differentiation to integration because the rules for differentiation are few and once they are known virtually any analytical function can be readily differentiated. Integration, although it is the inverse operation of differentiation, is much less loved because, depending upon one's familiarity, analytical integration involves searching through integral tables with no hope that an analytical form of the integral even exists.

Thus numerical integration is welcomed because it provides a simple and methodical procedure to evaluate integrals, without resorting to tables and regardless of the existence of an analytical form. That said, if an analytical form of the integral exists, it is preferable to use it. From a physical point of view, having an analytical function gives us insight into the parameters appearing within the function. From a professional point of view, we will be laughed at by our peers if we try to present work using numerical integration where an analytical form was readily apparent. Still, if there is a problem to be solved and no analytical integral in sight, numerical integration can, more often than not, come to the rescue.

6.2. Trapezoidal Rule

The Trapezoidal rule gets its name from the use of trapezoids to approximate integrals. Consider that you want to integrate a function, $f(x)$, from *a* to *b*. The trapezoidal rule says that the integral of that function can be approximated by a trapezoidal with a base of length (*b*-*a*) and sides of height $f(a)$ and $f(b)$. Graphically, the trapezoidal rule is represented in Figure 6.1.

The single-interval trapezoidal rule is expressed as

$$\int_a^b f(x)dx \approx \frac{1}{2}\big(f(a)+f(b)\big)(b-a) \tag{6.1}$$

The right hand side of equation (6.1) is the expression for the area of a trapezoid, shown in the Figure 6.1. Now, it is quite easy to imagine a case where the single-interval trapezoidal rule is going to give a terrible estimate. Consider the curve shown in Figure 6.2. (top). In order to increase the accuracy of the trapezoidal rule, one can approximate the integral by many smaller trapezoids. One can see in Figure 6.2. (bottom) that as the number of trapezoids increases, the area of the integral not accounted for by the trapezoidal rule decreases.

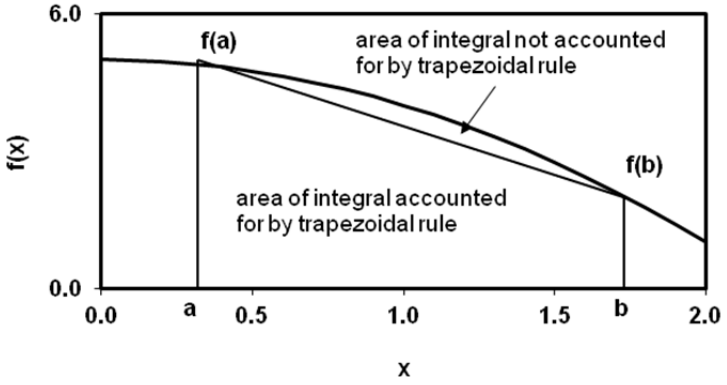

Figure 6.1. Schematic of trapezoidal rule for integration.

If we are integrating from *a* to *b* using *n* trapezoids (or intervals), then the base of each trapezoid (or discretization in the x-dimension) is

$$h = \frac{b-a}{n} \qquad (6.2)$$

The area of each of these smaller trapezoids, A_i, is

$$A_i = \frac{h}{2}\left(f(x_i) + f(x_{i+h})\right) \qquad (6.3)$$

where the position of each point, x_i, is given

$$x_i = a + (i-1) * h \qquad (6.4)$$

for i = 1 to n+1. We note that if there are n intervals, there must be n+1 points, with $x_1 = a$ and $x_{n+1}=b$.

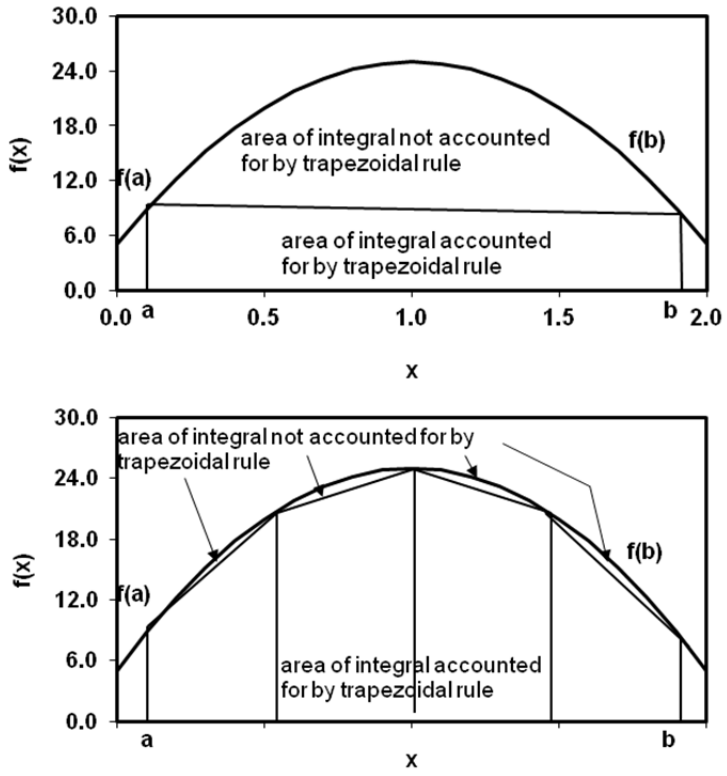

Figure 6.2. Accuracy of the trapezoidal rule improves as the number of trapezoids used increases.

The integral is given by the summation of the areas of all the trapezoids:

$$\int_a^b f(x)dx \approx \frac{h}{2}\sum_{i=1}^{n}\left(f(x_i)+f(x_{i+h})\right) \tag{6.5}$$

The quantity $f(x_i)$ appears twice in the summation of equation (6.5). It appears once as the left-hand-side of the trapezoid that forms the ith interval and it occurs once as the right-hand-side of the trapezoid that forms the i-1th interval. This is true of all $f(x_i)$ except the endpoints, $f(a)$ and $f(b)$. For these reasons, equation (6.5) can be algebraically manipulated to yield:

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(a)+f(b)+2\sum_{i=2}^{n} f(x_i)\right] \tag{6.6}$$

This is the most common form of the mutliple-interval trapezoidal rule. The accuracy of the trapezoidal rule increases as n increases. The trapezoidal rule is a first order method, which means two things. First, a first-order polynomial (a straight line) has been used to approximate the along each interval. Second, the error is proportional to the interval size to the first power. Halving the size of the interval halves the error. We postpone a comparative discussion of the accuracy of various methods until more methods have been introduced. Suffice it to say that the trapezoidal rule is not very accurate and we would prefer when possible to use higher order methods.

A MATLAB code which implements the trapezoidal rule is provided later in this chapter.

6.2. Second-Order Simpson's Rule

The Second-Order Simpson's Rule (often called the Simpson's 1/3 Rule although not in this book) is another technique used for numerical integration. A more accurate approach to integration involves the use of higher order polynomial approximating the integrand. The Second-Order Simpson's Rules uses a second-order (quadratic) polynomial.

The application of the 2nd order Simpson's rule is demonstrated graphically in Figure 6.3. The

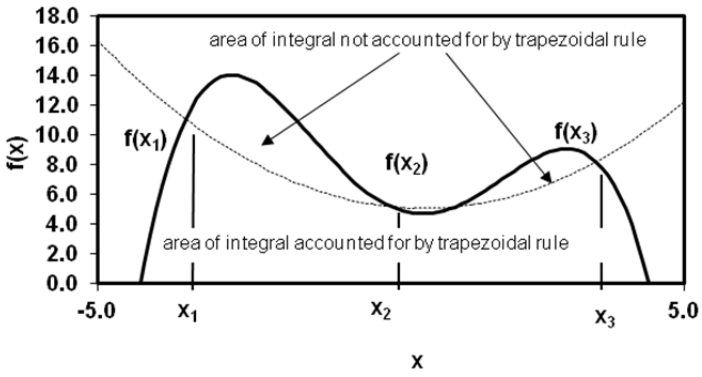

Figure 6.3. Schematic of second-order Simpson's rule for integration.

function is evaluated at three points (or equivalently across two intervals). A parabola is fit through those three points. Since there is a simple analytical formula for the integral of the parabola, the integration is then done analytically.

The derivation of the second-order Simpson's rule for numerical integration is an interesting application of linear algebra and regression skills that we already know from Chapters 1 and 2. Consider that we have three equally spaced points, x_1, x_2 and x_3. The x_2 and x_3 are related to x_1 by $x_2 = x_1 + \Delta x$ and $x_3 = x_1 + 2\Delta x$. We also know the function value evaluated at these three points, $f(x_1)$, $f(x_2)$, $f(x_3)$.

We are going to fit a second-order polynomial (parabola) to these three points. Since a parabola has three coefficients, we can perfectly fit the three points, $(x_1, f(x_1))$, $(x_2, f(x_2))$ and $(x_3, f(x_3))$. Substituting these three points into a second-order polynomial yields

$$
\begin{aligned}
c_2 x_1^2 + c_1 x_1 + c_0 - f(x_1) &= 0 \\
c_2 x_2^2 + c_1 x_2 + c_0 - f(x_2) &= 0 \\
c_2 x_3^2 + c_1 x_3 + c_0 - f(x_3) &= 0
\end{aligned}
$$

We have a system of three algebraic equations. Moreover, the equations are linear in the unknown coefficients. We can write these equations in matrix form, $\underline{\underline{A}}\,\underline{x} = \underline{b}$, where

$$
\underline{\underline{A}} = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix}, \qquad \underline{x} = \begin{bmatrix} c_2 \\ c_1 \\ c_0 \end{bmatrix}, \qquad \underline{b} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix}
$$

The solution to this system of equations is

$$
\underline{x} = \begin{bmatrix} c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} \dfrac{f(x_1) - 2f(x_2) + f(x_3)}{2\Delta x^2} \\ -\dfrac{f(x_1)\left[2x_1 + 3\Delta x\right] - f(x_2)\left[4x_1 + 4\Delta x\right] + f(x_3)\left[2x_1 + \Delta x\right]}{2\Delta x^2} \\ \dfrac{f(x_1)\left[x_1^2 + 3x_1\Delta x + 2\Delta x\right] - f(x_2)\left[2x_1^2 + 4x_1\Delta x\right] + f(x_3)\left[x_1^2 + x_1\Delta x\right]}{2\Delta x^2} \end{bmatrix}
$$

At this point we have the coefficients of our parabola. Now we want to integrate the parabola from x_1 to x_3.

$$I_{Simp} = \int_{x_1}^{x_3} \left(c_2 x^2 + c_1 x + c_0\right) dx = \left[\frac{c_2 x^3}{3} + \frac{c_1 x^2}{2} + c_0 x \right]_{x_1}^{x_3}$$

$$I_{Simp} = \left[\frac{c_2 x_3^3}{3} + \frac{c_1 x_3^2}{2} + c_0 x_3 \right] - \left[\frac{c_2 x_1^3}{3} + \frac{c_1 x_1^2}{2} + c_0 x_1 \right]$$

We substitute in the formulae for c_2, c_1, and c_0. We also substitute in $x_2 = x_1 + \Delta x$ and $x_3 = x_1 + 2\Delta x$. After a lot of messy but straightforward algebra, we find that the expression simplifies to:

$$I_{Simp} = \frac{\Delta x}{3}\left[f(x_1) + 4 f(x_2) + f(x_3) \right] \tag{6.7}$$

This is the second-order Simpson's rule where we have only 2 intervals (one parabola). If we divide the interval up into many intervals, then we have to sum each of the integrated intervals up. Just as was the case with the trapezoidal rule, the first and last points appear only once. Each interior point at the beginning or end of a parabola will be counted twice. Each point in the center of the parabola is counted once, but has a weighting factor of four. So, our final second-order Simpson's rule for *n*-intervals (*n*+1 points) is:

$$I_{Simp} = \frac{\Delta x}{3}\left[f(x_1) + 4 \sum_{\substack{i=2 \\ even}}^{n} f(x_2) + 2 \sum_{\substack{i=3 \\ odd}}^{n-1} f(x_2) + f(x_{n+1}) \right] \tag{6.8}$$

Again, as was the case for the Trapezoidal rule, the second-order Simpson's rule provides greater accuracy if the range is broken up into a number of intervals. For the the second-order Simpson's rule, the number of intervals, n, must be even because, as you can see in the above plot, each polynomial fit requires 2 intervals.

For the same number of intervals, the second-order Simpson's rule is more accurate than the trapezoidal rule because we used a high-order polynomial to fit the original function. Because the second-order Simpson's rule is second order, the error is proportional to the interval size to the second power. Halving the size of the interval reduces the error by a factor of four. We postpone a comparative discussion of the accuracy of various methods until more methods have been introduced.

A MATLAB code which implements the second-order Simpson's rule is provided later in this chapter.

6.3. Higher Order Simpson's Rules

The third-order Simpson's rule uses a third order (cubic) polynomial fit over three intervals to approximate the integral. The derivation of the method is precisely analogous to that for the second-order method. We present the result with proof. For a single application (three intervals or four points), the integral is

$$I_{Simp} = \frac{3\Delta x}{8}\left[f(x_1) + 3f(x_2) + 3f(x_3) + f(x_4)\right] \tag{6.9}$$

If we want to use many intervals, then we need to add up these individual components. We must also use a number of intervals that is a multiple of 3.

$$I_{Simp} = \frac{3\Delta x}{8}\left[f(x_1) + 3\sum_{i=2,5,8}^{n-1} f(x_2) + 3\sum_{i=3,6,9}^{n} f(x_2) + 2\sum_{i=4,7,10}^{n-2} f(x_2) + f(x_{n+1})\right] \tag{6.10}$$

The fourth-order Simpson's rule uses a fourth order (quartic) polynomial fit over four intervals to approximate the integral. For a single application (four intervals or five points), the integral is

$$I_{Simp} = \frac{2\Delta x}{45}\left[7f(x_1) + 32f(x_2) + 12f(x_3) + 32f(x_4) + 7f(x_5)\right] \tag{6.11}$$

If we want to use many intervals, then we need to add up these individual components. We must also use a number of intervals that is a multiple of 4.

$$I_{Simp} = \frac{2\Delta x}{45}\left[7f(x_1) + 32\sum_{i=2,6,10}^{n-2} f(x_2) + 12\sum_{i=3,7,11}^{n-1} f(x_2) + 32\sum_{i=4,8,12}^{n} f(x_2) + 14\sum_{i=5,9,13}^{n-3} f(x_2) + 7f(x_{n+1})\right] \tag{6.12}$$

MATLAB codes which implement the third-order and fourth-order Simpson's rule are provided later in this chapter.

6.5. Quadrature

Quadrature takes a slightly different approach to the numerical evaluation of integrals. Quadrature is based on the assumption that we can get a better estimate of the integral with fewer function evaluations if we use non-equally spaced points at which to evaluate the function. The determination of these points and the weighting coefficients that correspond to each data point follows a methodical procedure. We do not derive them here.

The integral for the nth order Gaussian Quadrature is given by:

$$I_{quad} = \sum_{i=1}^{n} c_i f(x_i) \tag{6.13}$$

The particular values of the weighting constants, *c*, and the points where we evaluate the function, *x*, can be taken from a table in any numerical methods text book.

The advantage of a quadrature technique is that it can be quite accurate for very few function evaluations. Thus it is much faster and if need to repeatedly evaluate integrals it is the method of choice. For the evaluation of a couple integrals, the Simpson's Rules are better because we know we can increase accuracy by increasing the number of intervals used.

MATLAB codes which implement Gaussian Quadrature for 2nd to 6th order are provided later in this chapter.

6.6. Example

We want to evaluate the change in entropy of methane for an isothermal expansion or compression. To describe the thermodynamic state of the fluid, we will rely on tried and true friend, the van der Waals equation of state.

$$p = \frac{RT}{V-b} - \frac{a}{V^2} \tag{4.16}$$

where P is pressure (Pa), T is temperature (K), V is molar volume (m^3/mol), R is the gas constant (8.314 J/mol/K = 8.314 Pa*m^3/mol/K), a is the van der Waal's attraction constant (.2303 Pa*m^6/mol^2 for methane) and *b* is the van der Waal's repulsion constant (4.306×10^{-5} m^3/mol for methane). The change in entropy for an isothermal expansion without phase change is

$$\Delta S = \int_{V_1}^{V_2} \left(\frac{\partial P}{\partial T} \right)_{V'} dV'$$

The partial derivative of the pressure with respect to temperature at constant molar volume for a van der Waal's gas is obtained from differentiating the van der Waals EOS.

$$\left(\frac{\partial P}{\partial T} \right)_V = \frac{R}{V-b}$$

so the change in entropy is

$$\Delta S = \int_{V_1}^{V_2} \frac{R}{V-b} dV'$$

We can analytically evaluate this integral so as to provide a basis of comparison for the accuracy of the various numerical techniques. The analytical integral is .

$$\Delta S = R \ln\left(\frac{V_2 - b}{V_1 - b}\right)$$

The pressure as a function of molar volume is shown in Figure 6.4. for van der Waals methane at 298 K. The partial derivative, $\left(\frac{\partial P}{\partial T}\right)_{\underline{V}}$, as a function of molar volume is also shown in Figure 6.4. for van der Waals methane at 298 K. The deriviative is the function we will need to numerically integrate. in order to obtain the entropy difference. It is a smooth, monotonically decreasing function in our range of interest.

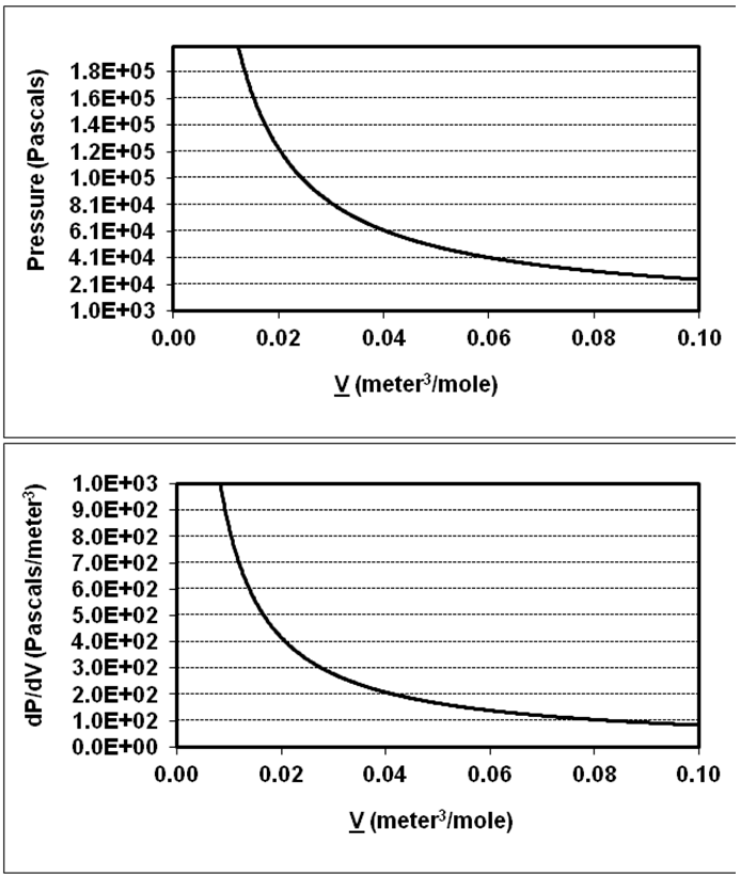

Figure 6.4. (top) Pressure as a function of molar volume at 298 K for van der Waals methane. (bottom) $\left(\frac{\partial P}{\partial T}\right)_{\underline{V}}$ as a function of molar volume at 298 K for van der Waals methane.

Let's expand the gas from 0.03 m^3/mol to 0.1 m^3/mol. We compare results using the analytical solution, and several numerical methods in Table 6.1. What is immediately obvious is that as we increase the number of intervals for a given order of method, we see a gradual increase in accuracy. In quadrature as we increase the order of the method, we decrease the error by an order of magnitude. Also, as we increase the order of a Simpson's-type method, we see a drastic increase in accuracy. For example it takes 100,000 intervals in the Trapezoidal rule to equal the accuracy of 100 intervals in the Simpson's Fourth Order Method. Quadrature is even more efficient. A 6-point quadrature equals the accuracy of 1000 trapezoidal intervals in this example.

Technique	# of intervals	$\Delta S(J/mol/K)$	percent error
analytical	-	10.0182	0.0
Trapezoidal	1	12.62476	2.60E+01
Trapezoidal	2	10.79212	7.73E+00
Trapezoidal	3	10.38034	3.61E+00
Trapezoidal	4	10.22639	2.08E+00
Trapezoidal	10	10.05242	3.42E-01
Trapezoidal	100	10.01854	3.44E-03
Trapezoidal	1000	10.01819	3.44E-05
Trapezoidal	10000	10.01819	3.44E-07
Trapezoidal	100000	10.01819	3.44E-09
Simpson's 2nd Order	2	10.18124	1.63E+00
Simpson's 2nd Order	4	10.03781	1.96E-01
Simpson's 2nd Order	10	10.01892	7.30E-03
Simpson's 2nd Order	100	10.01819	8.17E-07
Simpson's 2nd Order	1000	10.01819	8.18E-11
Simpson's 3rd Order	3	10.09979	8.14E-01
Simpson's 3rd Order	6	10.02738	9.18E-02
Simpson's 3rd Order	9	10.02040	2.21E-02
Simpson's 3rd Order	99	10.01819	1.91E-06
Simpson's 3rd Order	999	10.01819	1.85E-10
Simpson's 4th Order	4	10.02825	1.00E-01
Simpson's 4th Order	8	10.01869	4.96E-03
Simpson's 4th Order	100	10.01819	3.39E-09
Simpson's 4th Order	1000	10.01819	3.55E-14
Quadrature 2nd Order	2	9.91943	9.86E-01
Quadrature 3rd Order	3	10.00942	8.75E-02
Quadrature 4th Order	4	10.01743	7.63E-03
Quadrature 5th Order	5	10.01812	6.61E-04
Quadrature 6th Order	6	10.01819	5.70E-05
quad (MATLAB)	?	10.01828	9.20e-04
quad8 (MATLAB)	?	10.01819	3.36e-08

Table 6.1. Comparison of accuracy of various numerical integration methods.

6.7. Multidimensional Integrals

Integrals over an area or a volume are not uncommon in science and engineering. The most conceptually straight forward approach would be to extend the existing one-dimensional Simpson's and quadrature methods to multidimensional integrals through sequential application. Let us examine this application, for the simplest case, which involves the Trapezoidal rule for two-dimensional integration with fixed limits of integration. This essentially involves integrating a function over a rectangular area.

A two-dimensional integral with fixed limits of integration can be written as

$$I_{2D} = \int_{xo}^{xf}\int_{yo}^{yf} f(x,y)dydx = \int_{xo}^{xf} g(x)dx \tag{6.14}$$

where the integrand of the outermost integral, $g(x)$, is

$$g(x) = \int_{yo}^{yf} f(x,y)dy \tag{6.15}$$

This integrand has no dependence on *y* since that functionality has been integrated out. As a reminder, the 1-D trapezoidal rule using *n* intervals (*n*+1 points) is

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(a) + f(b) + 2\sum_{i=2}^{n} f(x_i) \right] \tag{6.6}$$

We apply the trapezoidal rule to the integral over y only first and substitute that into $g(x)$

$$g(x) = \int_{yo}^{yf} f(x,y)dy \approx \frac{h_y}{2}\left[f(x,y_o) + f(x,y_f) + 2\sum_{i=2}^{n_y} f(x,y_i) \right] \tag{6.16}$$

Substituting the discretized approximation for $g(x)$ in equation (6.16) into equation (6.14) yields

$$\int_{xo}^{xf}\int_{yo}^{yf} f(x,y)dydx \approx \int_{xo}^{xf} \frac{h_y}{2}\left[f(x,y_o) + f(x,y_f) + 2\sum_{i=2}^{n_y} f(x,y_i) \right] dx \tag{6.17}$$

Well, we can repeat the application of the trapezoidal rule:

$$I_{2D} \approx \frac{h_x}{2}\left\{\begin{array}{l} \dfrac{h_y}{2}\left[f(x_o,y_o) + f(x_o,y_f) + 2\sum_{i=2}^{n_y} f(x_o,y_i)\right] \\ + \dfrac{h_y}{2}\left[f(x_f,y_o) + f(x_f,y_f) + 2\sum_{i=2}^{n_y} f(x_f,y_i)\right] \\ + 2\sum_{j=2}^{n_x} \dfrac{h_y}{2}\left[f(x_j,y_o) + f(x_j,y_f) + 2\sum_{i=2}^{n_y} f(x_j,y_i)\right] \end{array}\right\} \tag{6.18}$$

Now we can simplify this as much as possible,

$$I_{2D} \approx \frac{h_x h_y}{4}\left\{\begin{array}{l} f(x_o,y_o) + f(x_o,y_f) + f(x_f,y_o) + f(x_f,y_f) + 4\sum_{i=2}^{n_y}\sum_{j=2}^{n_x} f(x_j,y_i) \\ + 2\sum_{i=2}^{n_y}\left[f(x_o,y_i) + f(x_f,y_i)\ \right] + 2\sum_{j=2}^{n_x}\left[f(x_j,y_o) + f(x_j,y_f)\ \right] \end{array}\right\} \tag{6.19}$$

If we add up the number of function evaluations, we can see that we have $(n_x+1)(n_y+1)$ function evaluations. If $n_x = n_y = n$, then we have $(n+1)^2$ function evaluations for a 2-D integral. By extension, if we need to evaluate an *m*-dimensional integral, then we will have $(n+1)^m$ function evaluations. In other words, the number of function evaluations scales exponentially with the dimensionality of the system. By observation we observe that there are four points with a weighting factor of one corresponding to the four corners of the area. There are four sums with a weighting factor of two corresponding to the four edges of the area. There is one double sum with a weighting factor of four corresponding to the interior of the area.

This simple procedure which culminates in the 2-D Trapezoidal rule can be applied to higher dimensions. For example, the 3-D Trapezoidal rule applied to an integral with fixed limits of integration yields a corresponding formula. This volume corresponds to a right parallelepiped. By observation we observe that there are eight points with a weighting factor of one corresponding to the eight vertices of the volume. There are twelve sums with a weighting factor of two corresponding to the twelve edges of the volume. There are six double sums with a weighting factor of four corresponding to the six faces of the volume. There is one triple sum with a weighting factor of eight corresponding to the interior of the volume.

$$I_{3D} = \frac{h_x h_y h_z}{2^3}\left[\begin{array}{l}
\left\{\begin{array}{l} f(x_o,y_o,z_o)+f(x_o,y_o,z_f)+f(x_o,y_f,z_o)+f(x_o,y_f,z_f) \\ +f(x_f,y_o,z_o)+f(x_f,y_o,z_f)+f(x_f,y_f,z_o)+f(x_f,y_f,z_f)\end{array}\right\} \\
+2\left\{\begin{array}{l} \sum_{i=2}^{n_z}\left[f(x_o,y_o,z_i)+f(x_o,y_f,z_i)+f(x_f,y_o,z_i)+f(x_f,y_f,z_i)\right] \\ +\sum_{j=2}^{n_y}\left[f(x_o,y_j,z_o)+f(x_o,y_j,z_f)+f(x_f,y_j,z_o)+f(x_f,y_j,z_f)\right] \\ +\sum_{k=2}^{nx}\left[f(x_k,y_o,z_o)+f(x_k,y_o,z_f)+f(x_k,y_f,z_o)+f(x_k,y_f,z_f)\right]\end{array}\right\} \\
+4\left\{\begin{array}{l} \sum_{j=1}^{n_y}\sum_{i=2}^{n_z}\left[f(x_o,y_j,z_i)+f(x_f,y_j,z_i)\right] \\ +\sum_{k=2}^{nx}\sum_{i=2}^{n_z}\left[f(x_k,y_o,z_i)+f(x_k,y_f,z_i)\right] \\ +\sum_{k=2}^{nx}\sum_{j=1}^{n_y}\left[f(x_k,y_j,z_o)+f(x_k,y_j,z_f)\right]\end{array}\right\} \\
+8\sum_{k=2}^{nx}\sum_{j=1}^{n_y}\sum_{i=2}^{n_z} f(x_k,y_j,z_i)
\end{array}\right] \quad (6.20)$$

This is the explicit form of the trapezoidal rule applied in 3-dimensions, when the limits of integration are constant

This simple procedure which culminates in the multidimensional Trapezoidal rule can be applied to higher order methods. For example, the two-dimensional Simpon's second order method applied to an integral with constant limits of integration can be derived in an analogous manner and is presented in equation (6.20). The simplified final expression involves terms with weighting factors of one (the corners of the area), weighting factors of two (odd-numbered nodes on the edges of the area), weighting factors of four (even-numbered nodes on the edges of the area and odd-odd double sums in the interior), weighting factors of eight (odd-even double sums in the interior) and weighting factors of sixteen (even-even double sums in the interior).

For any of these methods, if the boundary of the area over which the integration occurs is not regular, simple functions like that given in equation (6.19), (6.20) and (6.21) cannot be written. However, the same concept can still be applied. Integration over *y* can be performed, generating a one-dimensional function $g(x)$, which can then be integrated.

For high dimensional integrals with complicated geometries, completely different methods of numerical integration are often invoked, including Monte Carlo integration, which randomly samples the function. Such methods are beyond the scope of this introductory text.

$$I_{2D} \approx \frac{h_x h_y}{9}\left\{\begin{array}{l} f(x_o,y_o)+f(x_o,y_f)+f(x_f,y_o)+f(x_f,y_f) \\ +2\left(\sum_{i=3,5,7}^{n_y-2}\left[f(x_o,y_i)+f(x_f,y_i)\right]+\sum_{j=3,5,7}^{n_x-2}\left[f(x_j,y_o)+f(x_j,y_f)\right]\right) \\ +4\left(\sum_{i=2,4,6}^{n_y-1}\left[f(x_o,y_i)+f(x_f,y_i)\right]+\sum_{j=2,4,6}^{n_x-1}\left[f(x_j,y_o)+f(x_j,y_f)\right]\right) \\ +8\left(\sum_{j=2,4,6}^{n_x-1}\sum_{i=3,5,7}^{n_y-2}f(x_j,y_i)+\sum_{j=3,5,7}^{n_x-2}\sum_{i=2,4,6}^{n_y-1}f(x_j,y_i)\right) \\ +4\sum_{j=3,5,7}^{n_x-2}\sum_{i=3,5,7}^{n_y-2}f(x_j,y_i)+16\sum_{j=2,4,6}^{n_x-1}\sum_{i=2,4,6}^{n_y-1}f(x_j,y_i) \end{array}\right\} \qquad (6.21)$$

6.8. Subroutine Codes

In this section, we provide a routine for implementing the various numerical integration methods described above. Note that these codes correspond to the theory and notation exactly as laid out in this book. These codes do not contain extensive error checking, which would complicate the coding and defeat their purpose as learning tools. That said, these codes work and can be used to solve problems.

As before, on the course website, two entirely equivalent versions of this code are provided and are titled *code.m* and *code_short.m*. The short version is presented here. The longer version, containing instructions and serving more as a learning tool, is not presented here. The numerical mechanics of the two versions of the code are identical.

Code 6.1. Trapezoidal Rule (trapezoidal_short)

```
function integral = trapezoidal_short(a,b,nintervals);
dx = (b-a)/nintervals;
npoints = nintervals + 1;
x_vec = [a:dx:b];
integral = funkeval(x_vec(1));
for i = 2:1:nintervals
   integral = integral + 2*funkeval(x_vec(i));
end
integral = integral + funkeval(x_vec(npoints));
integral = 0.5*dx*integral;
fprintf(1,'\nUsing the Trapezoidal method \n');
fprintf(1,'to integrate from %f to %f with %i nintervals,\n',a,b,nintervals);
fprintf(1,'the integral is %e \n \n',integral);
```

```
function f = funkeval(x)
f = x^2 + 5 - sin(x);
```

An example of using trapezoidal_short is given below.

```
» integral = trapezoidal_short(0.03,0.1,100);

Using the Trapezoidal method
to integrate from 0.030000 to 0.100000 with 100 nintervals,
the integral is 3.457785e-001
```

Code 6.2. Simpson's Second Order Rule (simpson2_short)

```
function integral = simpson2_short(a,b,nintervals);
if (mod(nintervals,2) ~= 0)
   fprintf('Simpsons 2nd method requires an even number of intervals.\n');
else
   dx = (b-a)/nintervals;
   npoints = nintervals + 1;
   x_vec = [a:dx:b];
   integral_first = funkeval(x_vec(1));
   integral_last = funkeval(x_vec(npoints));
   integral_4 = 0.0;
   for i = 2:2:nintervals
         integral_4 = integral_4 + funkeval(x_vec(i));
   end
   integral_2 = 0.0;
   for i = 3:2:nintervals-1
         integral_2 = integral_2 + funkeval(x_vec(i));
   end
   integral = integral_first + integral_last + 4.0*integral_4 + 2.0*integral_2;
   integral = dx/3*integral;
   fprintf(1,'\nUsing the Simpsons Second Order method \n');
   fprintf(1,'to integrate from %f to %f with %i nintervals,\n',
a,b,nintervals);
   fprintf(1,'the integral is %e \n \n',integral);
end

function f = funkeval(x)
f = x^2 + 5 - sin(x);
```

An example of using simpson2_short is given below.

```
» integral = simpson2_short(0.0,1.0,100)

Using the Simpsons Second Order method
to integrate from 0.000000 to 1.000000 with 100 nintervals,
the integral is 4.873636e+000
```

Code 6.3. Simpson's Third Order Rule (simpson3_short)

```
function integral = simpson3_short(a,b,nintervals);
if (mod(nintervals,3) ~= 0)
      fprintf('Simpsons 3rd Order method requires a # of intervals that is a
multiple of 3.\n');
else
   dx = (b-a)/nintervals;
   npoints = nintervals + 1;
   x_vec = [a:dx:b];
   integral_first = funkeval(x_vec(1));
   integral_last = funkeval(x_vec(npoints));
   integral_3a = 0.0;
   for i = 2:3:nintervals-1
         integral_3a = integral_3a + funkeval(x_vec(i));
   end
   integral_3b = 0.0;
   for i = 3:3:nintervals
         integral_3b = integral_3b + funkeval(x_vec(i));
   end
   integral_2 = 0.0;
   for i = 4:3:nintervals-2
         integral_2 = integral_2 + funkeval(x_vec(i));
   end
   integral = integral_first + integral_last + 3.0*integral_3a ...
      + 3.0*integral_3b + 2.0*integral_2;
   integral = 3.0*dx/8.0*integral;
   fprintf(1,'\nUsing the Simpsons Third Order method \n');
   fprintf(1,'to integrate from %f to %f with %i
nintervals,\n',a,b,nintervals);
   fprintf(1,'the integral is %e \n \n',integral);
end

function f = funkeval(x)
f = x^2 + 5 - sin(x);
```

An example of using simpson3_short is given below.

```
» integral = simpson3_short(0.0,1.0,99)

Using the Simpsons Third Order method
to integrate from 0.000000 to 1.000000 with 99 nintervals,
the integral is 4.873636e+000
```

Code 6.4. Simpson's Fourth Order Rule (simpson4_short)

```
function integral = simpson4_short(a,b,nintervals);
if (mod(nintervals,4) ~= 0)
      fprintf('Simpsons 4th Order method requires a # of intervals that is a
multiple of 4.\n');
else
```

```
   dx = (b-a)/nintervals;
   npoints = nintervals + 1;
   x_vec = [a:dx:b];
   integral_first = funkeval(x_vec(1));
   integral_last = funkeval(x_vec(npoints));
   integral_32a = 0.0;
   for i = 2:4:nintervals-2
         integral_32a = integral_32a + funkeval(x_vec(i));
   end
   integral_32b = 0.0;
   for i = 4:4:nintervals
         integral_32b = integral_32b + funkeval(x_vec(i));
   end
   integral_12 = 0.0;
   for i = 3:4:nintervals-1
         integral_12 = integral_12 + funkeval(x_vec(i));
   end
   integral_14 = 0.0;
   for i = 5:4:nintervals-3
         integral_14 = integral_14 + funkeval(x_vec(i));
   end
   integral = 7.0*integral_first + 7.0*integral_last + 32.0*integral_32a ...
      + 32.0*integral_32b + 12.0*integral_12 + 14.0*integral_14;
   integral = 2.0*dx/45.0*integral;
   fprintf(1,'\nUsing the Simpsons Fourth Order method \n');
   fprintf(1,'to integrate from %f to %f with %i
nintervals,\n',a,b,nintervals);
   fprintf(1,'the integral is %e \n \n',integral);
end

function f = funkeval(x)
f = x^2 + 5 - sin(x);
```

An example of using simpson4_short is given below.

```
» integral = simpson4_short(0.0,1.0,100)

Using the Simpsons Fourth Order method
to integrate from 0.000000 to 1.000000 with 100 nintervals,
the integral is 4.873636e+000
```

Code 6.5. Gaussian Quadrature (gaussquad_short)

```
function integral = gaussquad_short(a,b,norder);
if (norder < 2 | norder > 6)
   fprintf('This code only works for order between 2 and 6\n');
else
   a0 = 0.5*(b+a);
   a1 = 0.5*(b-a);
   if (norder == 2)
      c(1) = 1.0;
```

```
   c(2) = c(1);
   x_table(1) = -0.577350269;
   x_table(2) = -x_table(1);
elseif (norder == 3)
   c(1) = 0.555555556;
   c(2) = 0.888888889;
   c(3) = c(1);
   x_table(1) = -0.774596669;
   x_table(2) = 0.0;
   x_table(3) = -x_table(1);
elseif (norder == 4)
   c(1) = 0.347854845;
   c(2) = 0.652145155;
   c(3) = c(2);
   c(4) = c(1);
   x_table(1) = -0.861136312;
   x_table(2) = -0.339981044;
   x_table(3) = -x_table(2);
   x_table(4) = -x_table(1);
elseif (norder == 5)
   c(1) = 0.236926885;
   c(2) = 0.478628670;
   c(3) = 0.568888889;
   c(4) = c(2);
   c(5) = c(1);
   x_table(1) = -0.906179846;
   x_table(2) = -0.538469310;
   x_table(3) = 0.0;
   x_table(4) = -x_table(2);
   x_table(5) = -x_table(1);
elseif (norder == 6)
   c(1) = 0.171324492;
   c(2) = 0.360761573;
   c(3) = 0.467913935;
   c(4) = c(3);
   c(5) = c(2);
   c(6) = c(1);
   x_table(1) = -0.932469514;
   x_table(2) = -0.661209386;
   x_table(3) = -0.238619186;
   x_table(4) = -x_table(3);
   x_table(5) = -x_table(2);
   x_table(6) = -x_table(1);
end
integral = 0.0;
for i = 1:1:norder
   x(i) = a0 + a1*x_table(i);
   f(i) = funkeval(x(i));
   integral = integral + c(i)*f(i);
end
integral = integral*a1;
fprintf(1,'\nUsing %i order Gaussian Quadrature \n', norder);
fprintf(1,'to integrate from %f to %f \n',a,b);
```

```
   fprintf(1,'the integral is %e \n \n',integral);
end

function f = funkeval(x)
R = 8.314;
b = 4.306e-5;
f = R/(x-b);
```

An example of using guassquad_short is given below.

```
» integral = gaussquad_short(0.03,0.1,4)

Using 4 order Gaussian Quadrature
to integrate from 0.030000 to 0.100000
the integral is 1.001743e+001
```

6.9. Problems

Problems are located on course website.