Lecture 39,40,42 - Numerical Integration

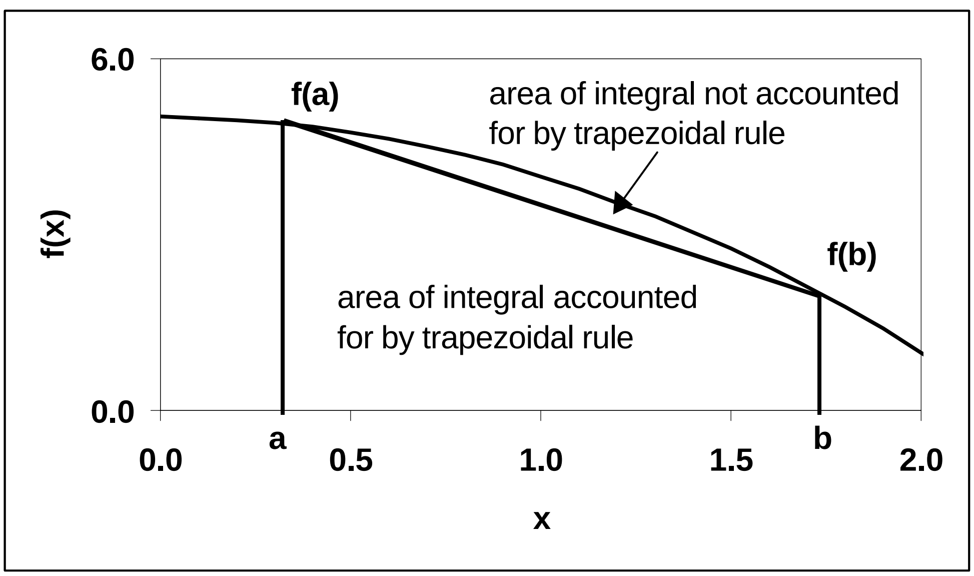
Table of Contents

39.1 Why is it important to be able to numerically integrate equations?

There are many instances in the engineering and the sciences where it is necessary to evaluate integrals. One case where you may require numerical integration occurs when the integrand may be of a functional form that you do not know how to integrate analytically. In this case, your only recourse is to numerically evaluate the integral. A second case where you may require numerical integration occurs when you need to integrate a function given by data points. In this case, there is no analytical solution to the integral and again you require numerical methods.

39.2 Trapezoidal Rule

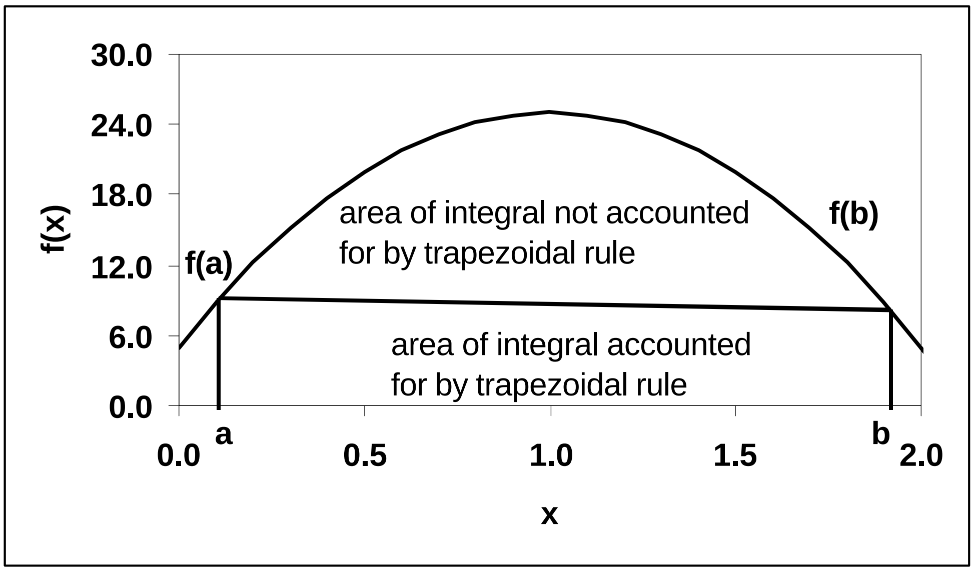
The Trapezoidal rule gets it's name from the use of trapezoids to approximate integrals. Consider that you want to integrate a function, $f(x)$, from a to b. The trapezoidal rule says that the integral of that function can be approximated by a trapezoidal with a base of length (b-a) and sides of height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



In terms of equations the, single-interval trapezoidal rule is expressed as

$$\int_a^b f(x)dx \approx \frac{1}{2}\left(f(a)+f(b)\right)(b-a) \tag{39.1}$$

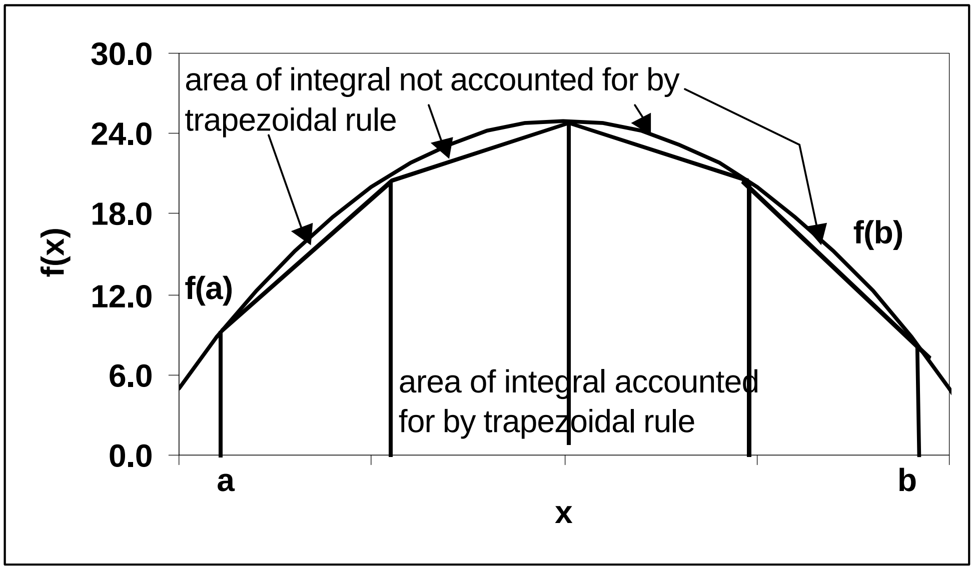
The right hand side of equation (39.1) is the expression for the area of a trapezoid, shown in the figure above. Now, it is quite easy to imagine a case where the single-interval trapezoidal rule is going to give a terrible estimate. Consider the following plot:



The method that is used to obtain a better estimate of the integral using the trapezoidal rule is to break up the range from a to b into n smaller intervals, each of size h where

$$h = \frac{b-a}{n} \tag{39.2}$$

Graphically, this is depicted below for n = 4



Visually, one can detect that even breaking the range into 4 intervals has substantially increased the accuracy of the trapezoidal rule. Also note that each interval is a trapezoid. The area of each of these trapezoids, A_i, is

$$A_i = \frac{h}{2}\left(f(x_i) + f(x_{i+h})\right) \tag{39.3}$$

where the position of the left side of the i[th] trapezoid is x_i, given by a linear interpolation formula between a to b for i = 1 to n intervals

$$x_i = a + (i-1) * h \tag{39.4}$$

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

$$\int_a^b f(x)dx \approx \frac{h}{2}\sum_{i=1}^{n}\left(f(x_i) + f(x_{i+h})\right) \tag{39.5}$$

The quantity $f(x_i)$ appears twice in the summation of equation (39.5). It appears once as the left-hand-side of the trapezoid that forms the i[th] interval and it occurs once as the right-hand-side of the trapezoid that forms the i-1[th] interval. This is true of all $f(x_i)$ except the endpoints, $f(a)$ and $f(b)$. For these reasons, equation (39.5) can be algebraically manipulated to yield:

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(a) + f(b) + 2\sum_{i=2}^{n} f(x_i)\right] \tag{39.6}$$

This is the most common form of the mutliple-interval trapezoidal rule. The accuracy of the trapezoidal rule increases as n increases.

Here is a MATLAB code that will implement the trapezoidal rule. This code is located in a file called trapezoidal.m. It is executed at the command line prompt by typing:

>> trapezoidal(a,b,nintervals)

where a and b are the lower and upper limits of integration. nintervals is the number of intervals. The function you wish to integrate is listed as the last line of the code. In this sample, f(x) = x.

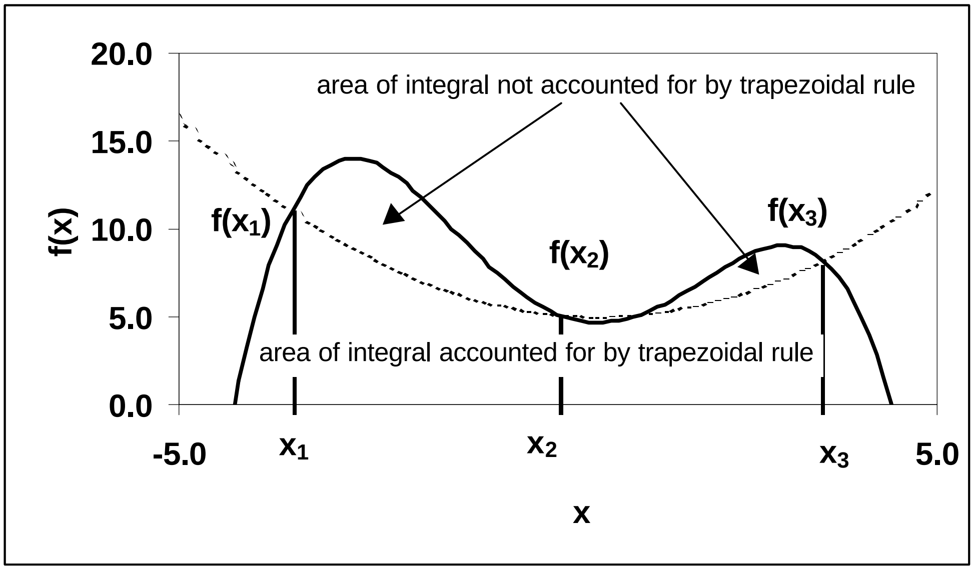
```
%
%  Trapezoidal Method
%
function integral = trapezoidal(a,b,nintervals);
dx = (b-a)/nintervals;
npoints = nintervals + 1;
x_vec = [a:dx:b];
integral = funkeval(x_vec(1));
for i = 2:1:nintervals
   integral = integral + 2*funkeval(x_vec(i));
end
integral = integral + funkeval(x_vec(npoints));
integral = 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;
```

39.3 Simpson's 1/3 Rule

Simpson's 1/3 Rule is another technique used for numerical integration. When we used the single-interval trapezoidal rule to estimate the integral of $f(x)$ over the range of a to b, we drew a straight line from the point $(a, f(a))$ to $(b, f(b))$. A more accurate approach might me to increase the level of the polynomial approximating the curve from linear (a polynomial of order one, as used in the trapezoidal rule) to quadratic (a polynomial of order two, as used in Simpson's 1/3 rule).

The application of Simpson's 1/3 rule to a curve is shown graphically below.



The parabolic curve which approximates the function matches the function at 3 points, the 2 endpoints and the centerpoint. (In the trapezoidal rule, the approximating curve was a line which matched the function only at the end-points.)

Derivation of the Simpson's 1/3 Rule for Numerical Integration.

We have three points, x_1, x_2 and x_3. x_2 and x_3 are related to x_1 by

$$x_2 = x_1 + \Delta x$$
$$x_3 = x_1 + 2\Delta x$$

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 parabola to these three points. The general equation of a parabola is a quadratic polynomial. Each of the three points must fit on the parabola. (Recall from the regression analysis that, if we have three points, we can fit them perfectly with the three coefficients of a quadratic polynomial.)

$$eqn_1 = ax_1^2 + bx_1 + c - f(x_1)$$
$$eqn_2 = ax_2^2 + bx_2 + c - f(x_2)$$
$$eqn_3 = ax_3^2 + bx_3 + c - f(x_3)$$

So we have three unknowns: a, b, and c. In fact, we have a system of three linear equations and three unknowns. So the equations are of the form:

$$\underline{\underline{A}}\,\underline{x} = \underline{b}$$

where

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

The solution to this system of equations is:

$$\underline{x} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \dfrac{f(x_1) - 2f(x_2) + f(x_3)}{2\Delta x^2} \\ -\dfrac{f(x_1)\left[2x_1 + 3\,\Delta x\right] - f(x_2)\left[4\,x_1 + 4\,\Delta x\right] + f(x_3)\left[2\,x_1 + \Delta x\right]}{2\Delta x^2} \\ \dfrac{f(x_1)\left[x_2^1 + 3x_1\Delta x + 2\,\Delta x\right] - f(x_2)\left[2x_2^1 + 4x_1\Delta x\right] + f(x_3)\left[x_2^1 + x_1\Delta x\right]}{2\Delta x^2} \end{bmatrix}$$

At this point we have the coefficients of our parabola.

Now we want to integrate the parabola from x_1 to x_3.

$$I_{Simp} = \int_{x_1}^{x_3} \left(ax^2 + bx + c\right) dx$$

$$I_{Simp} = \left[\frac{ax^3}{3} + \frac{bx^2}{2} + cx\right]_{x_1}^{x_3}$$

$$I_{Simp} = \left[\frac{ax_3^3}{3} + \frac{bx_3^2}{2} + cx_3\right] - \left[\frac{ax_1^3}{3} + \frac{bx_1^2}{2} + cx_1\right]$$

We substitute in the formulae for a, b, and c. We also substitute in $x_3 = x_1 + 2\Delta x$. After a lot of extremely messy but simple algebra, we find that the expression simplifies to:

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

This is Simpson's 1/3 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 point 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 Simpson's 1/3 rule formula for n-intervals (n+1 points) is:

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

Again, as was the case for the Trapezoidal rule, Simpson's 1/3 rule provides greater accuracy if the range is broken up into a number of intervals. For the Simpson's 1/3 rule, the number of intervals, n, must be even because, as you can see in the above plot, each curve fit requires 2 intervals.

The Simpson's 1/3 rule requires an even number of intervals, because it needs two intervals per parabola.
For the same number of intervals, the Simpson's 1/3 rule is more accurate than the Trapezoidal rule because we used a high-order polynomial to fit the original function.

Here is a MATLAB code that will implement Simpson's 1/3 rule. This code is located in a file called simpson2.m because this is Simpson's second-order method. (Remember, we fit the curve with a second-order (quadratic) polynomial.) It is executed at the command line prompt by typing:

>> simpson2(a,b,nintervals)

where a and b are the lower and upper limits of integration. nintervals is the number of intervals. The function you wish to integrate is listed as the last line of the code. In this sample, f(x) = x.

```
%
%  Simpson's Second Order Method (the 1/3 Rule)
%
function integral = simpson2(a,b,nintervals);
if (mod(nintervals,2) ~= 0)
   fprintf('Simpsons Second Order 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;
```

39.4 Simpson's Higher Order Methods

What we have presented above are the lowest two polynomial approximations. The trapezoidal rule used a first-order (linear) curve to model the function and the Simpson's 1/3 Rule used a second-order (quadratic) curve to model the function. If we wanted to, we could derive equations for any arbitrary higher order polynomial.

We could repeat the derivation for Simpson's rule where we used a third-order (cubic) polynomial to obtain the solution.

The result for a single curve (three intervals) is

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

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.

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

Here is a MATLAB code that will implement Simpson's Third Order Method. This code is located in a file called simpson3.m . It is executed at the command line prompt by typing:

```
>> simpson3(a,b,nintervals)
```

where a and b are the lower and upper limits of integration. nintervals is the number of intervals. The function you wish to integrate is listed as the last line of the code. In this sample, f(x) = x.

```
%
%  Simpson's Third Order Method
%
function integral = simpson3(a,b,nintervals);
if (mod(nintervals,3) ~= 0)
    fprintf('Simpsons Third 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;
```

We could repeat the derivation for Simpson's rule where we used a fourth-order (quartic) polynomial to obtain the solution.

The result for a single curve (four intervals) is

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

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.

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

Here is a MATLAB code that will implement Simpson's Fourth Order Method. This code is located in a file called simpson4.m . It is executed at the command line prompt by typing:

>> simpson4(a,b,nintervals)

where a and b are the lower and upper limits of integration. nintervals is the number of intervals. The function you wish to integrate is listed as the last line of the code. In this sample, f(x) = x.

```
%
%  Simpson's Fourth Order Method
%
function integral = simpson4(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;
```

39.5 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:

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

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 the quadrature 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.

Here, we provide a code which performs Gaussian Quadrature for 2^{nd} to 6^{th} order. This code is located in a file called gaussquad.m . It is executed at the command line prompt by typing:

>> gaussquad(a,b,norder)

where a and b are the lower and upper limits of integration. norder is the order of the approximation. The function you wish to integrate is listed as the last line of the code. In this sample, f(x) = R/(x-b).

You will see that most of the code is simply a table of weighting coefficients and x values.

```
%
%  Gaussian Quadrature
%
function integral = gaussquad(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);
```

39.6 Example: Comparison of Methods

We want to evaluate the change in entropy of methane for an isothermal expansion or compression. We will use the van der Waal's equation of state.

The van der Waal's equation of state is:

$$P = \frac{RT}{\underline{V} - b} - \frac{a}{\underline{V}^2} \tag{39.9}$$

where P is pressure (Pa), T is temperature (K), $\underline{V}$ 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 (4.306e-5 m^3/mol for methane). The change in entropy for an isothermal expansion without phase change is

$$\Delta S = \int_{\underline{V}_1}^{\underline{V}_2} \left(\frac{\partial P}{\partial T}\right)_{\underline{V}'} d\underline{V}' \tag{39.10}$$

The partial derivative of the pressure with respect to temperature at constant molar volume for a van der Waal's gas is (from equation 39.9)

$$\left(\frac{\partial P}{\partial T}\right)_{\underline{V}} = \frac{R}{\underline{V} - b} \tag{39.11}$$

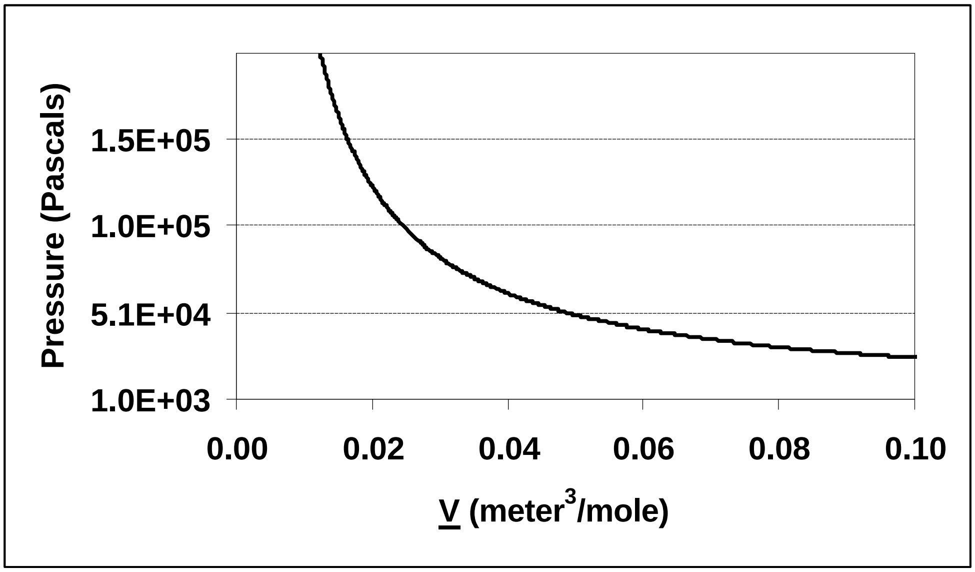
so the change in entropy is

$$\Delta S = \int_{\underline{V}_1}^{\underline{V}_2} \left(\frac{R}{\underline{V}' - b}\right) d\underline{V}' \tag{39.12}$$

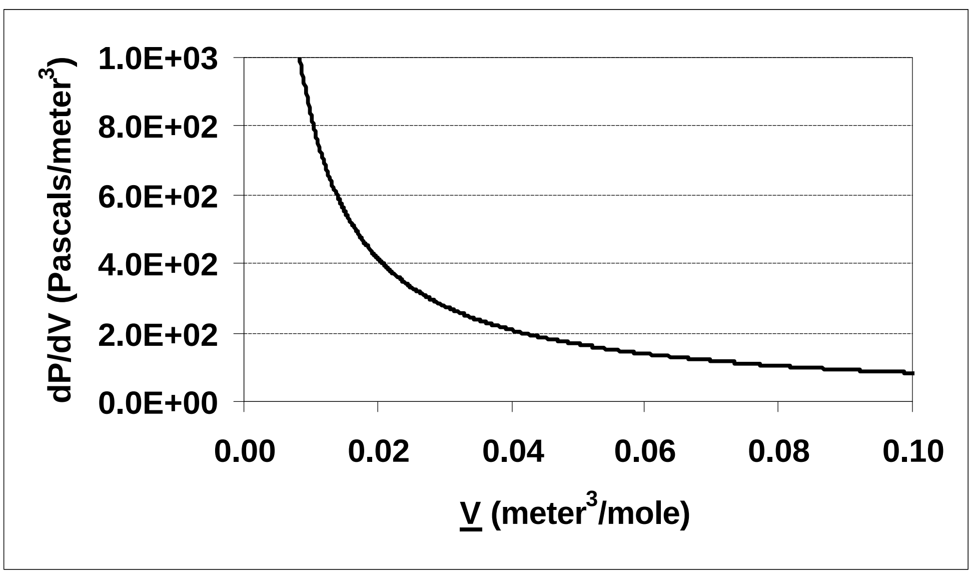
We can analytically evaluate this integral so as to provide a basis of comparison with our numerical integrals.

$$\Delta S = R \ln\left(\frac{\underline{V}_2 - b}{\underline{V}_1 - b}\right) \tag{39.13}$$

The pressure as a function of molar volume is shown below for van der Waal's methane at 298 K.



$\left(\frac{\partial P}{\partial T}\right)_{\underline{V}}$ as a function of molar volume is shown below for van der Waal's methane at 298 K. This is the function we will need to numerically integrate.



Let's expand the gas from 0.03 m^3/mol to .1 m^3/mol. We compare results using the analytical solution, and several numerical methods. We see that as we increase the number of intervals, we increase the accuracy of our solution. We also see that as we increase the order of the method, the accuracy increases.

Technique	# of intervals	$\Delta S(J/mol/K)$	percent error
analytical (equation 39.13)	-	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 1/3	2	10.18124	1.63E+00
Simpson's 1/3	4	10.03781	1.96E-01
Simpson's 1/3	10	10.01892	7.30E-03
Simpson's 1/3	100	10.01819	8.17E-07
Simpson's 1/3	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
quad (MATLAB)	?	10.01828	9.20e-04
quad8 (MATLAB)	?	10.01819	3.36e-08

39.7 Numerical Integration - MATLAB

MATLAB has two built-in numerical integration routine called quad and quad8, which use quadrature. I have built a routine called "integrate.m" which will integrate either an analytical function in the file 'fn.m' or numerical data in the file 'file.anything.dat' using your choice of trapezoidal, Simpson's 1/3 Rule, or MATLAB's quad.

The arguments of the integrate.m routine can be seen by moving to the directory where the integrate.m file is located and typing `help integrate`

» help integrate

integrate(type,n,a,b,m,'fname')
This script performs one dimension integration of a function or of data

type = 1, integrate a function in the file 'fn.m'
type = 2, integrate data in the file 'fname'
n = number of intervals
a = lower bound of integration
b = upper bound of integration
m = integration type

    if m = 1, then use trapezoidal rule
    if m = 2, then use Simpson's 1/3 rule
    if m = 3, then use Gaussian quadrature

fname is the name of the data file

    For this program, the fname must be 'file.anything.dat' !!!
    (You need single quotes around the filename.)

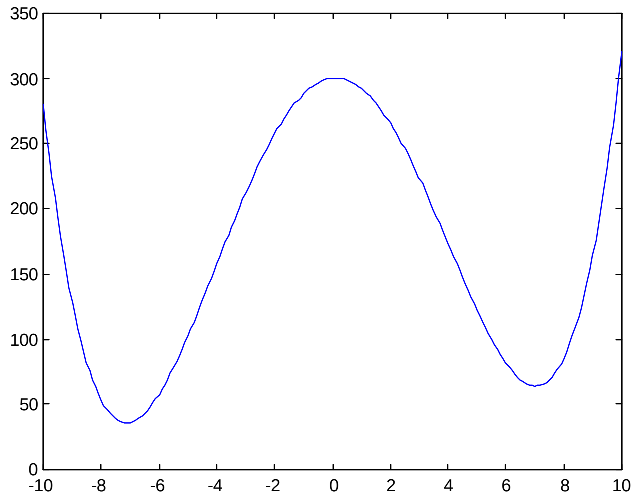
For type =1, fname is not used.
(But something must be entered anyway.)

For type = 2, a, b, and m inputs are not used.
(But something must be entered anyway.)
The input file must have 2 columns.
The first column is the dependent variable given in evenly spaced intervals.
The second column gives values of the function to be integrated.
There must be n+1 rows in the file.

Example #2.

```
b0 = 300.0;
b1 = 2.0;
b2 = -10.0;
b3 = 0.0;
b4 = +0.1;
p = b0 + b1*v + b2*v^2 + b3*v^3 + b4*v^4;
```

which looks like



Find $\int_{-10}^{10} Pdv$

Exact analytical solution:

$$\int_{-10}^{10} Pdv = b0 * v + \frac{b1 * v^2}{2} + \frac{b2 * v^3}{3} + \frac{b3 * v^4}{4} + \frac{b4 * v^5}{5}\Bigg|_{-10}^{10}$$

$$\int_{-10}^{10} Pdv = 1766.667 - -1566.667 = 3333.333$$

The data in the following table was generated in MATLAB using the integrate.m routine with arguments along the lines of:

integrate(1,100,-10,10,2,'fn')

Technique	n	$\Delta S(J/mol/K)$	percent error
analytical (equation 39.13)	-	3333.333	0.0
Trapezoidal	1	6000	80.00
Trapezoidal	2	6000	80.00
Trapezoidal	3	4683.1	40.49
Trapezoidal	4	4125	23.75
Trapezoidal	10	3465.6	3.97
Trapezoidal	100	3334.7	0.04
Trapezoidal	1000	3333.3	0.0
Trapezoidal	10000	3333.3	0.0
Trapezoidal	100000	3333.3	0.0
Simpson's 1/3	2	6000	80.00
Simpson's 1/3	4	3500	5.00
Simpson's 1/3	10	3337.6	0.13
Simpson's 1/3	100	3333.3	0.0
quad (MATLAB)	?	3333.3	0.0

39.8 Numerical Integration of Data

Example #3. Integrating Data

Integrating data is no different from integrating a function. Here the function has already been evaluated for us. We can use the trapezoidal rule or the Simpson's 1/3 rule. The number of intervals is defined as one less than the number of data points.

Consider the data in the file 'file.test.dat'

f(x)	x
1.00	0.0
1.01	0.1
1.04	0.2
1.09	0.3
1.16	0.4
1.25	0.5
1.36	0.6
1.49	0.7
1.64	0.8
1.81	0.9
2.00	1.0

Integrate this data using the trapezoidal rule from x = 0.0 to 1.0
There are eleven data points and thus ten intervals.
The MATLAB command with the integrate.m routine is used as follows:

```
integrate(2,10,0,1,1,'file.test.dat')
```

```
y = 1.3333
```

Hopefully, it is clear that we are performing the same operations regardless of whether we have a function or data to integrate. When we have data, we ought to think of it as having the function already evaluated for us.

39.9 MATLAB - GUIs

A GUI (pronounced gooey) is a graphical user interface, which makes using codes simpler. There is a GUI for numerical integration, developed by Dr. Keffer, located at:
http://clausius.engr.utk.edu/webresource/index.html .

You have to download and unzip the GUI. Then, when you are in the directory where you extracted the files, you type:

```
>>integrate_gui
```

at the command line prompt to start the GUI.

I give no additional instructions here because a good GUI is self-explanatory. If you understand the methods in this lecture packet, you will have no problem using the GUI.