NUMERICAL INTEGRATION

Numerical integration is a mathematical technique used to solve integrals. It can be done without taking calculus. Students who have not completed calculus should not be intimidated by this material. Most of it deals with calculating areas under curves, which should not be difficult for those who have not completed calculus. That being said, it is a good idea to learn how numerical integration is different from analytical integration.

In the first semester of calculus, the average student will deal with limits and taking derivatives. The most important definition learned in the first calculus class is the following:

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

This is the definition of a derivative. Graphically, it is the slope exactly at a point. An analytical solution (or exact solution) exists if this limit can be evaluated. The student will spend a large portion of their calculus course learning/memorizing many of these limits. For example, many students will memorize that the derivative of x^2 is $2x$ or the derivative of e^x is equal to itself.

Once this material has been mastered the student moves on to Calculus II where they do a significant amount of integration. Integration involves taking the anti-derivative of an equation, which is basically differentiation in reverse. They will see the following equation very often:

$$y = \int f(x)dx$$

This is an indefinite integral. The answer will be the anti-derivative plus a constant. The student will spend a significant amount of time learning anti-derivatives of many functions. For example, the anti-derivative of $2x$ is x^2 and the anti-derivative of e^x is itself. In engineering, indefinite integrals are rarely used. Instead a definite integral is used. A definite integral includes upper and lower bounds for the integration and does not require the constant mentioned above. It would look like:

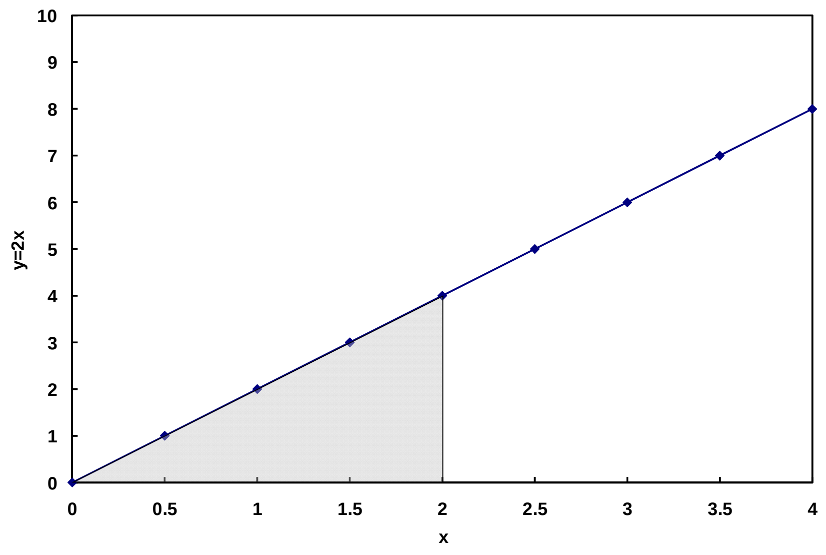
$$y = \int_{x_l}^{x_u} f(x)dx$$

In this equation, x_u is the upper bound of integration and x_l is the lower bound of integration. The solution to this integral is the anti-derivative of f evaluated at x_u minus the anti-derivative of f evaluated at x_l. If both bounds are numbers (rather than variables), the solution yields a number value.

Take a look at the example of *f(x) = 2x*. If you were to take the integral of *2x* from 0 to 2, where 0 is the lower bound and 2 is the upper bound you would get the following:

$$y = \int_0^2 2x dx = x_u^2 - x_l^2 = 2^2 - 0^2 = 4$$

An interesting trend arises, if you were to plot the equation *2x*:



If you take the area under the line between 0 and 2, it would be the area of a triangle. This triangle will have an area equal to one-half the base times the height, which is ½ times 2 times 4. This product is exactly the same as the integral from 0 to 2. Now consider if we were to integrate from 1 to 3. Solved analytically, the integral would be:

$$y = \int_1^3 2x dx = x_u^2 - x_l^2 = 3^2 - 1^2 = 8$$

Now if we go back to our plot of *2x*, the area under the line from 1 to 3 is equal to a large triangle minus the smaller triangle:

Area = ½*3*6 – ½*1*2 = 9 – 1 = 8

Again, the area under the line is equal to the integral. THIS IS NOT A COINCIDENCE. The integral is in fact equal to the area under a curve over some lower and upper bound. Given this information, we can calculate integrals without knowledge of how to do an anti-derivative, because we know the integral is the area under the curve.

The technique we will use is called numerical integration. Although it sounds like a silver bullet to Calculus II, numerical integration does not yield an exact solution for most functions. Calculating the area under a curve is not an exact science, so there is almost always some error in numerical integration. Also, there are probably well over a dozen different ways to numerically integrate a function. Each method will have a different accuracy for different functions.

One question that should pop up in the reader's mind is: If numerical integration is not as accurate as the anti-derivative, why would we ever use it? This is a great question and there are two common scenarios where numerical integration is important. The first situation is when you have data. It is possible to integrate data. If the data cannot be fit to a function, then there is no function to take the anti-derivative of. The other scenario where numerical integration is useful is when the function does not have an anti-derivative. Despite what you may have learned or will learn in Calculus II, every function does not have an anti-derivative. Since numerical integration calculates the area under a curve, any function can be integrated in this fashion. An example of each case will be given below.

NUMERICAL INTEGRATION OF DATA

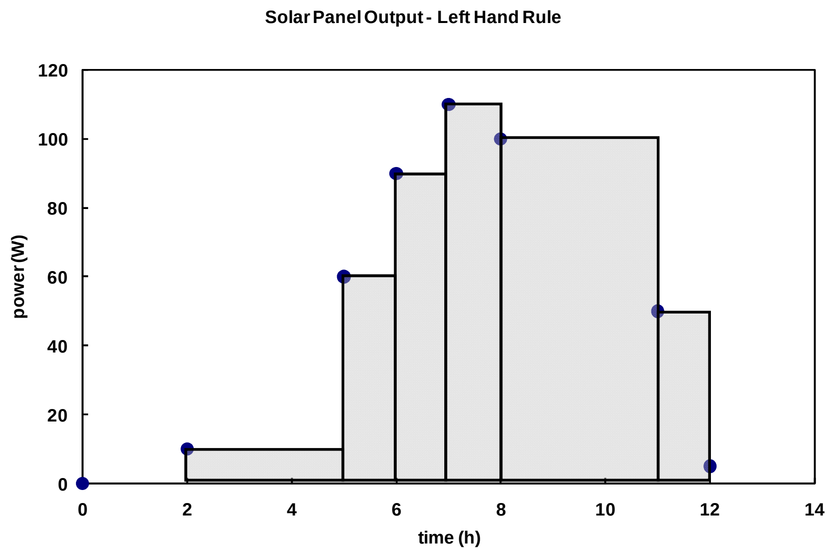
As an example take the amount of power generated by a solar panel over the course of a day. The instantaneous power (watts, W) of a solar panel can be measured at different times during the day. Below is a table of the data, where the first row is the time of day, the second row is the time after 6 am, and the third row is the instantaneous power generation:

Time of Day	6 am	8 am	11 am	12 pm	1 pm	2 pm	5 pm	6 pm
Time (hours)	0	2	5	6	7	8	11	12
Power (watts)	0	10	60	90	110	100	50	5

Now the question is: How much energy in watt-hours (Wh) can we generate per day using a solar panel? An amount of energy can be found by multiplying the power times the elapsed time. If the power were constant throughout the day, the task would be trivial. Since it is not, we will need to do a numerical integration. We can make assumptions on how the rate changes in between data points, which will affect the numerical integration technique we use. The first two techniques, the left hand rule and the right hand rule, are not generally accepted, but they do help to introduce the notion of error in the integration.

Left Hand Rule

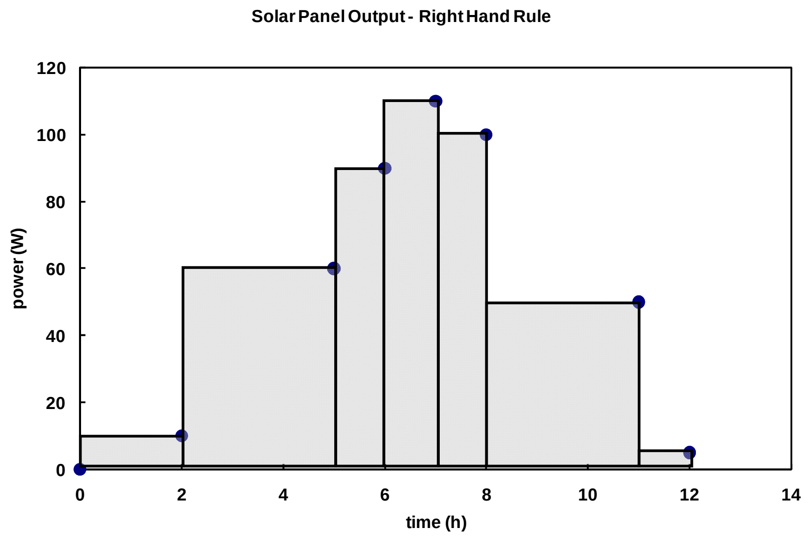
Using the left hand rule, one assumes that the value in between the first point and the second point is equal to the first point. For our data above, the first interval would have a power of 0 watts, the second interval would have a power of 10 watts, etc. until the next to last data point is reached This is easier to see graphically:



The total energy for the day is found by adding up the areas of all of the rectangles. The area of the first interval would be 2h*0W or 0Wh, the second interval would be 3h*10W or 30Wh, etc., and the final interval would be 1h*50W or 50Wh. The total value is 640 Wh. It is called the left hand rule, because the height of each rectangle is set to the data point on the left hand side of the interval.

Right Hand Rule

The right hand rule is similar to the left hand rule except instead of taking the data point on the left as the height, the data point on the right is taken. Again, this is easier to see graphically.

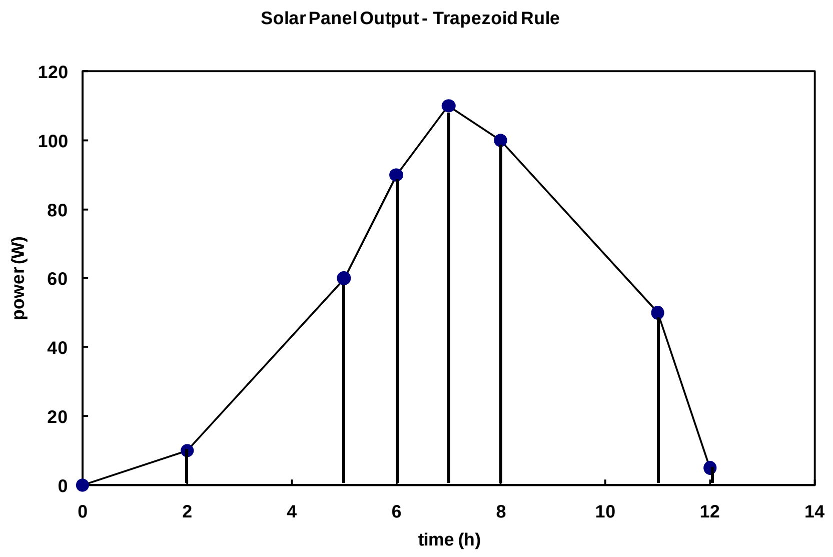


By adding up the area under the rectangles, the total energy for the day can be calculated. The area of the first interval would be 2h*10W or 20Wh, the second interval would be 3h*60W or 180Wh, etc., and the final interval would be 1h*5W or 5Wh. The total value is 655 Wh.

In general, the right hand and left hand rules are not used in numerical integration. They are the simplest to implement, but they lead to the largest errors especially when a function is given. Both of these rules assume that the values (power in our case) is constant in between data points, then instantaneously rises or falls to the next value. Since the power generated by the solar panel is related to the intensity of the sun, it is unreasonable to assume that the intensity of the sunlight changes instantaneously as we take a data point.

Trapezoid Rule

The trapezoid rule assumes that the data in between data points linearly rises or falls to the next value. This is the assumption that is generally accepted as the most basic, but sufficiently accurate. The plot below shows graphically what areas are being calculated:



The areas are calculated by finding the average value of each interval and multiplying by the interval width. The first interval would be (10+0)/2 * 2h =10Wh, the second interval would be (10+60)/2 * 3h or 105 Wh, etc., and the last interval would be (50+5)/2 * 1 h or 27.5 Wh. The total is 647.5 Wh.

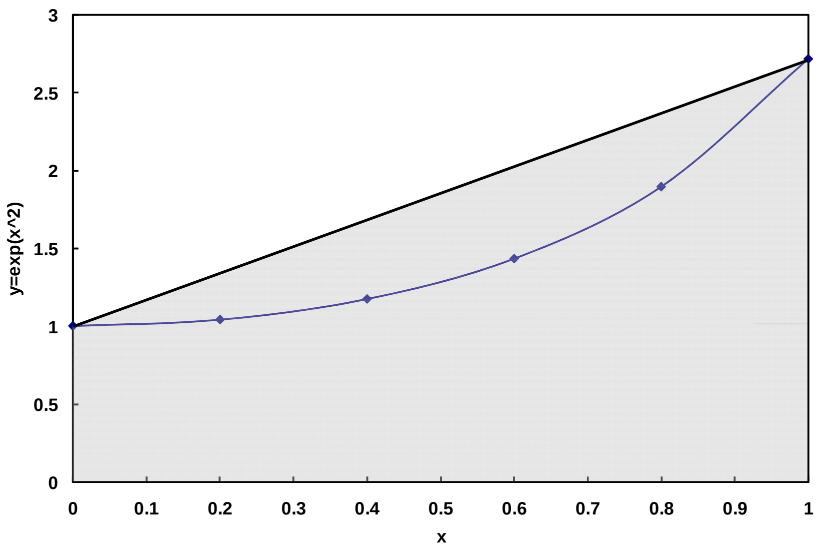
Before we discuss MATLAB, let’s look at an example of a function that does not have an anti-derivative.

NUMERICALLY INTEGRATING A FUNCTION

It is important to keep in mind that any function can be numerically integrated even if an analytical solution exists. An example of a function that does not have an anti-derivative is the exponential function below:

$$f(x) = e^{x^2}$$

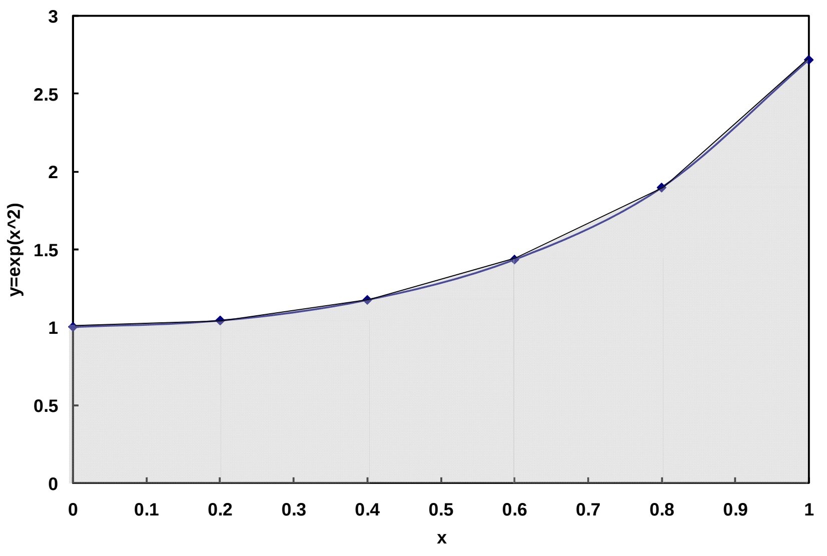
The trapezoid rule can be used to integrate this function as well. It is possible to integrate using one trapezoid. However, the trapezoid will clearly have some error, which is easy to see from the plot. Below is an example of integrating the function from 0 to 1.



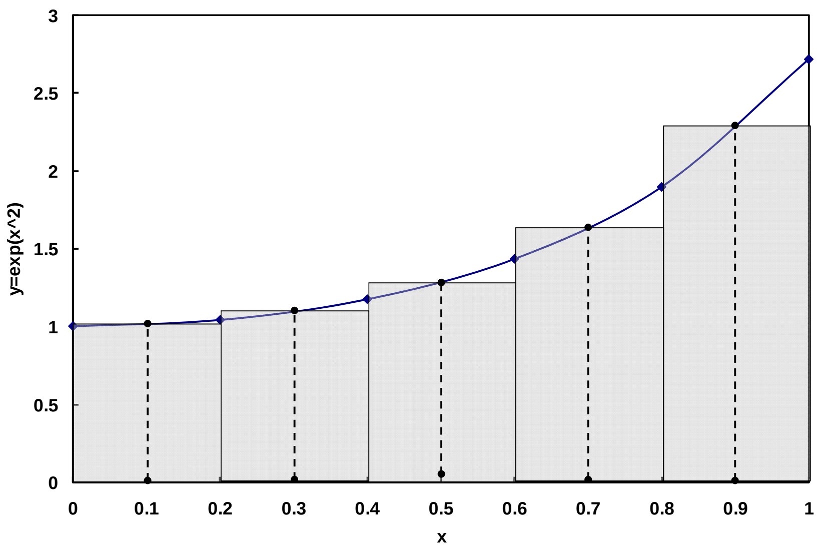
One way to decrease the error from this technique is to break the interval up into several intervals and perform the trapezoid rule on each interval similar to what was done with the data. Typically, the step size or interval width will be the same for all of the intervals.

It is important to take note that as the interval width goes to zero the number of intervals goes to infinity. The limit as the number of intervals goes to infinity yields the same solution as the analytical solution. As the number of intervals increases, the use of a computer program to calculate the integral makes more and more sense. In fact, any program that solves integrals will do it in this fashion and not use an actual analytical solution.

If the interval were broken up into 5 intervals, the error will decrease as shown by the plot below:



Another technique for integration is to use the midpoint of each interval to calculate the height rather than the average value of the beginning and end of the interval. At the end of this tutorial, you will be asked how you would program such a method. The midpoint method is shown below graphically for the same function.



One final technique that is well accepted for functions is called Simpson's Rule. There are several variations of Simpson's Rule. The one we will look at calculates the area of an interval as:

$$AI = (x_{i+1} - x_i)\frac{f(x_i) + 4f(x_{mid}) + f(x_{i+1})}{6}$$

In this equation, x_{i+1} is the upper bound of the interval, x_i is the lower bound of the interval, and x_{mid} is the midpoint of the interval. This equation treats the interval like a parabolic function rather than a trapezoid. The derivation of Simpson's Rule will be covered sufficiently in Calculus II, so it will be left out of this tutorial.

PROGRAMMING NUMERICAL METHODS IN MATLAB

Integrating Data with the Trapezoid Rule

For our set of data above, we can first write down pseudocode to do the integration. It might look something like:

```
Input times
Input power outputs
Calculate the average power for interval 1
Calculate the time length for interval 1
Calculate the energy for interval 1
Calculate the average power for interval 2
Calculate the time length for interval 2
Calculate the energy for interval 2
REPEAT for all intervals
Calculate the average power for interval 7
Calculate the time length for interval 7
Calculate the energy for interval 7
Sum the energies
Output the total
```

The pseudocode can be translated into a MATLAB function m-file. One possible result would look like:

```
function [tote]=energy(t,W) %the function call has the inputs and outputs
len=length(t); %the length of either input could be used, because they are the same
tote=0; %the total must be initialized
for ii=1:len-1 %There is one less interval than the number of data points, hence len-1
Wa=(W(ii+1)+W(ii))/2; %this is the average value of the data
Int=t(ii+1)-t(ii); %this is the time interval
AI=Wa*int; %this is the total energy for the interval or the area of that interval
totE=totE+AI; %this adds up all of the intervals
end
```

That is the entire m-file. After it has been saved, it can be run in the command window, by defining the inputs and then calling the function:

```
>>t=[0 2 5 6 7 8 11 12];
>>W=[0 10 60 90 110 100 50 5];
>>[totE]=energy(t,W)
totE=
647.5
```

Integrating a Function with the Trapezoid Rule

The major difference between integrating data and a function is that the number of steps/intervals to integrate a function is usually defined by the user. For data, the frequency of data points will determine the size of the interval. To integrate our exponential function, our pseudocode could look like:

```
Input lower bound
Input upper bound
Input interval size or number of intervals
Calculate the area of interval 1
REPEAT
Calculate the area of the final interval
Output the total area
```

There are a few changes from the data integration pseudocode. The pseudocode has been simplified; in the previous pseudocode, the calculation of area was actually three lines. In this example we will shorten to one line. The other difference is that we now have a third input, which can be the number of intervals or the size of the interval. Since the number of intervals will determine the accuracy of the method, it is often preferred to have the number of intervals as an input. However, this will require another calculation, because the interval width will be used in the *for* loop, not the number of intervals. The opposite is true if the interval width is entered. Except for special cases or advanced integration, the interval width is usually constant throughout the integration.

If we were to translate the above pseudocode into MATLAB specifying the width rather than the number of intervals, the function m-file to numerically integrate our exponential function would look like:

```
function [I]=intex2(xl,xu,wid)
% xl is the lower bound, xu is the upper bound, and wid is the interval width
I=0; % the integral needs to be initialized
for ii=xl:wid:xu-wid
AI=((exp((ii+wid)^2)+exp(ii^2))/2)*wid;
I=I+AI;
end
```

In only six lines of code, we can integrate our equation. By setting the width properly, we could solve for 1 million intervals, meaning we have calculated the area of 1 million trapezoids in only 6 lines of code.

With theoretically infinite numerical methods to choose from, this tutorial could go on indefinitely. It is important to note that the only difference between numerical methods is how they calculate the area of a single interval, AI. Given a numerical method with its equation to calculate AI, the student should be able to use that method to integrate the equation.

ERROR IN NUMERICAL INTEGRATION OF A FUNCTION

In Calculus II, error in different numerical integration techniques will be discussed. The student will learn how much more accurate Simpson's Rule is over the Trapezoid Rule. If a program as we wrote above were to calculate the integral using 100 000 or 1 million intervals, the error would be very small, i.e. orders of magnitude less than 1%. That being said any of the numerical methods discussed will be sufficient to solve the problems found in the class with a sufficient accuracy. However, it is still valuable to calculate the error of a numerical method to get a ballpark figure. The most common way to quantify the error is to subtract the result of the numerical solution from the analytical solution and divide by the analytical solution:

$$error = \frac{\left|S_{analytical} - S_{numerical}\right|}{S_{analytical}} * 100\%$$

$S_{analytical}$ is the analytical solution and $S_{numerical}$ is the numerical solution. The absolute value is taken to account equally for over and under estimates of the solution.

The common question is the following: if we are using numerical integration on a function that does not have an analytical solution, how do we use the analytical solution to calculate the error? The way to get around this is to use the numerical method on a similar function that does have an analytical solution. Then the amount of error can be related to the number of intervals for each numerical method. This is a simple way to calculate the error. More exact methods to calculate the error require knowledge of Calculus I & II, so they will be left out of this tutorial.

EXAMPLE

Write an m-file to calculate the integral of the function below using Simpson's Rule and to calculate the error:

$$f(x) = \frac{1}{x}$$

The exact solution for this integrated from x_l to x_u is the natural logarithm of x_u divided by x_l.

We can use the same code we used above except we will change the name and we will change the calculation of the area of an interval. We must change the calculation of the area, because we are changing the function and the numerical method. The m-file would look like:

```
function [I,err]=int1x(xl,xu,wid)
% xl is the lower bound, xu is the upper bound, and wid is the interval width
I=0; % the integral needs to be initialized
for ii=xl:wid:xu-wid
AI=((1/ii)+4*(1/(ii+wid/2))+(1/(ii+wid)))/6*wid;
I=I+AI;
end
anI=log(xu/xl); % This is the analytical solution, where log is the natural log
err=abs(anI-I)/anI*100;  %The abs function calculates the absolute value
```

This function can be run in the command window to calculate the integral over any range.

DO IT YOURSELF

1) Modify the program for the exponential of x^2 function to integrate using Simpson’s Rule or the Midpoint Rule.
2) Modify either of the codes above, so that the number of intervals is an input rather than the width of the interval.