Chapter 4

In-Class Activity Day 3: Highlights of Single-Variable Calculus: Integrals Review

4.1 Integrals

Additional resources for this subsection:

- Khan Academy: Integrals as Riemann sums

Consider an explicit function $y = f(x)$. The area of the region below the curve defined by $y = f(x)$, above the line $y = 0$, and between the lines $x = a$ and $x = b$, is the definite integral of f from $x = a$ to $x = b$,

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i)\Delta x_i \tag{4.1}$$

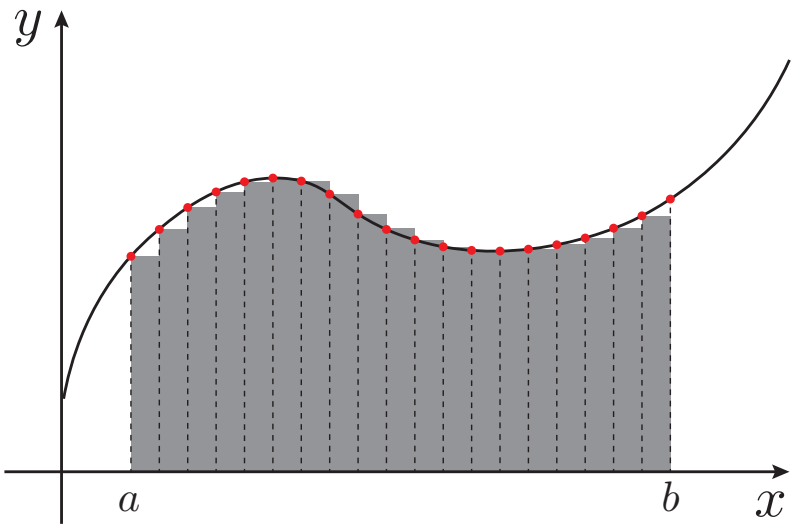


Figure 4.1: The integral as a limit. As the width of the partition gets smaller and smaller, the sum of the areas of the rectangles with heights equal to the values of the function at the endpoint of the interval, approaches the area under the curve (presuming this limit exists, which is the case for Riemann integrable functions).

4.1.1 Numerical Integration

In the same way that we were able to use the slope of the secant to approximate the derivative of a function, we can use a finite Riemann sum to approximate the integral of a function. Specifically:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_i)\Delta x_i \tag{4.2}$$

where n is some large (but finite) number. Computing this approximation by hand is pretty tedious. In MATLAB, however, the process is relatively straightforward. For example, let's use a Riemann sum to numerically approximate the following integral

$$F(x) = \int_0^x 3\bar{x}^2 d\bar{x} \tag{4.3}$$

for $x \in [0, 5]$. Note that we will be evaluating the **definite** integral of $f(\bar{x}) = 3\bar{x}^2$ (i.e. we are integrating f over a **specified** interval), where the integration bounds are given by $\bar{x} \in [0, x]$. We'll begin by defining the range ($x \in [0, 5]$), and the number of rectangles to use ($n = 100$).

```
%define the range of x
%and the number of rectangles to use
a = 0; b = 5; n = 100;
```

Once a, b, and n have been defined, we can use the linspace function to generate the values of x_i that we need to compute the Riemann sum approximation. X = linspace(a,b,k) computes a vector of k evenly spaced values, where $X(1) = a$ and $X(k) = b$. This corresponds to $k - 1$ rectangles (since we need to account for the left and right sides of the first and last rectangles). As such, if we want to use n rectangles in our Riemann sum, we will need to a range of $n + 1$ values (we will be ignoring the last value):

```
%define a set of n+1 evenly spaced values of x from a to b
%where x_vals(1) = a, and x_vals(n+1) = b
x_vals = linspace(a,b,n+1);
```

We can then evaluate $f(x_i) = 3x_i^2$ for each value of x_i in our range:

```
%evaluate f(x)=3*x^2 for each x in x_vals
y_vals = 3*x_vals.^2;
%note the use of .^ (element-wise exponentiation)
%instead of ^ (matrix exponentiation)
```

If the values of x_i are evenly spaced, then the width of each rectangle, $\Delta x = x_{i+1} - x_i$ is given by:

$$\Delta x = \frac{b - a}{n} \tag{4.4}$$

This can be computed in MATLAB as follows:

```
%compute the width of each rectangle
dx = x_vals(2)-x_vals(1);
%if you use n+1 points for linspace,
%you should find that dx = (b-a)/n
```

One nice way to find the Riemann sum is to use MATLAB's **cumulative sum** function (cumsum) which generates the running total of elements in a vector:

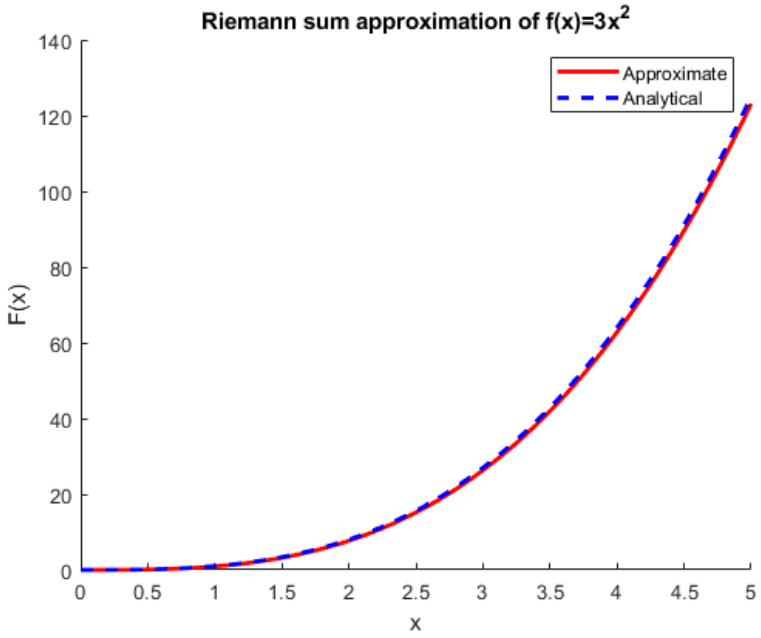
$$\text{cumsum}([1, 1, 1, 2, -1]) = [1, 2, 3, 5, 4] \tag{4.5}$$

For the purpose of this example, we implement this as follows:

```
%compute the cumulative sum of y_vals
temp_vals = cumsum(y_vals);

%approximate the integral of f(x) using the Riemann sum
F_vals = [0,dx*temp_vals(1:end-1)];
```

Note that you could also use MATLAB's trapz and cumtrapz functions here instead of sum and cumsum. To evaluate the performance of the approximation, let's generate a plot comparing it with the analytical solution, $F(x) = x^3$:

```
%make it so calls to plot do not erase the plot beforehand
hold on

%plot the numerical and analytical solutions
plot(x_vals,F_vals,'r','linewidth',2);
plot(x_vals,x_vals.^3,'b--','linewidth',2);

%make axis labels, title and legend
xlabel('x');
ylabel('F(x)');
title('Riemann sum approximation of f(x)=3x^2');
h = legend('Approximate','Analytical');
```



From the plot, it looks like our approximation was actually pretty good! The true power of this method is that we can use it to evaluate the integral of any function, even the many functions whose integrals lack a closed-form expression (in other words, it is impossible to integrate the function by hand).

Exercise 4.1

Use MATLAB's to numerically integrate the following functions. For each function, generate a plot comparing **three** different approximations of its integral over the specified interval (one for $n = 5$, one for $n = 20$, and one for $n = 1000$, where n is the number of rectangles in the Riemann sum).

1. $$F(x) = \int_{-1.5}^{x} 2\sin(\pi\bar{x})d\bar{x}, \quad x \in [-1.5, 1.5] \tag{4.6}$$

2. $$F(x) = \int_{-5}^{x} 1.5e^{.2\bar{x}}d\bar{x} \quad x \in [-5, 5] \tag{4.7}$$

3. $$F(x) = \int_{0}^{x} \cos(\bar{x}) - 3\bar{x}^2 + 2\ d\bar{x} \quad x \in [0, 6] \tag{4.8}$$

4.1.2 Symbolic Integration

Just as we used MATLAB's built-in computer-algebra engine to compute derivatives, we can also use it to compute integrals. To take the symbolic integral of a function, use the int command:

```
%computes the indefinite integral of y
F = int(y);
```

For example, the following block of code computes the integral of $f(x) = 3x^2$:

```
%defines x and y as symbolic variables
syms x y;
%set y to the symbolic expression 3*x^2
y=3*x^2;
%computes the symbolic indefinite integral of 3*x^2
F = int(y);
```

Note that the indefinite integral generated by the int command drops the integration constant. Now that we have our symbolic expression for $F(x) = \int f(x)dx$, we can use the subs command to evaluate $F(x)$ along a range of desired values:

```
%define a list of x values
x_vals = -1:.01:1;
%compute F(x) at each of the values in x_vals
F_vals_symbolic = subs(F,x_vals)
```

Note that the output of subs are still 'symbolic'. To convert the result into the equivalent numerical form, we need to use the double command:

```
%convert from symbolic to numeric
F_vals = double(F_vals_symbolic);
```

The int command can also be used to compute definite integrals. For example, to compute the symbolic expression for

$$F(x) = \int_a^b 3x^2 dx \tag{4.9}$$

we would use the following block of code:

```
%defines a and b as symbolic variables
syms a b;
%compute the definite integral on the interval [a,b]
F_definite1 = int(y,a,b);
```

For the specific case of $a = 2$ and $b = 7$:

```
%compute the definite integral on the interval [2,7]
F_definite2 = int(y,2,7);
```

Note that the output of int is always a symbolic data type, so if you want to convert it to a numeric data type, you will need to use the double command.

Exercise 4.2

Use MATLAB's symbolic toolbox to compute the integrals of the following functions. For each function, generate a plot comparing the symbolic integral with the approximate integrals that you computed in the previous exercise.

1. $$F(x) = \int_{-1.5}^{x} 2\sin(\pi\bar{x})d\bar{x}, \quad x \in [-1.5, 1.5] \tag{4.10}$$

2. $$F(x) = \int_{-5}^{x} 1.5e^{.2\bar{x}}d\bar{x} \quad x \in [-5, 5] \tag{4.11}$$

3. $$F(x) = \int_{0}^{x} \cos(\bar{x}) - 3\bar{x}^2 + 2\ d\bar{x} \quad x \in [0, 6] \tag{4.12}$$

4.2 Application: Tracking the Neato's Heading

Let's apply what we've learned about numerical differentiation and integration to measure the heading of the Neato as a function of time. Take a look at the following video which depicts the Neato being driven around MAC 128 via remote-control. Encoders in the wheels of the Neato allow us to track how much each wheel has turned. The encoder data generated by the Neato during that video has been provided as the file 'neato_data_day_assignment03.mat', which can be found on the assignment page.

We can use MATLAB's load function to import the data from the .mat file. In this case, all the data has been in a single matrix named 'recorded_data'. We can use the following block of code to import the data:

```
%path where the .mat file is stored
my_path = 'C:\Users\otaylor\Documents\MATLAB\';
%name of the .mat file we want to load
fname = 'neato_data_day_assignment03.mat';
%load the variable recorded_data from the .mat file
%and save it into the variable neato_data
neato_data = load([my_path,fname],'recorded_data').recorded_data;
```

Here, the variable 'neato_data' is an $n \times 5$ matrix, where each column corresponds to the following five data streams:

$$[t, x_l, x_r, v_l, v_r] \tag{4.13}$$

where:

- t is the time-stamp (in seconds) of the measured data-point.
- x_l and x_r are the measured linear travel (in meters) of the left and right wheels respectively.
- v_l and v_r are the commanded (not actual/measured) linear velocities (in meters/second) of the left and right wheel respectively.

We can unpack the 'neato_data' matrix into different vectors that we can analyze separately:

```
%unpack neato_data into 5 separate vectors
tlist =  neato_data(:,1);
left_wheel_encoder_list = neato_data(:,2);
right_wheel_encoder_list = neato_data(:,3);
left_wheel_vel_input_list = neato_data(:,4);
right_wheel_vel_input_list = neato_data(:,5);
```

Exercise 4.3

Your task is to use the encoder measurements to track the Neato's heading (relative to its initial orientation) as a function of time.

1. Generate a plot comparing the encoder measurements for both wheels as a function of time.
2. Use the finite difference formula (the secant method approximation) to approximate the linear velocity of each wheel as a function of time. Generate a plot comparing the measured velocity with the commanded velocity (as functions of time) for each wheel. Keep in mind that the data points are not necessarily evenly spaced in time.
3. The rotation rate of the Neato in the horizontal plane (in rad/sec) is given by:

   $$\dot{\theta} = \frac{v_r - v_l}{D} \tag{4.14}$$

   where v_l and v_r left/right linear velocities of the wheels, and D is the distance between the two wheels ($D = 24$ cm $= .24$ m), and $\theta > 0$ corresponds to a counterclockwise rotation (w/respect to the initial orientation). Generate a plot of $\dot{\theta}$ as a function of time.
4. We can numerically integrate $\dot{\theta}$ to approximate θ. Use the numerical integration technique that you implemented in the previous exercise to generate a plot of θ as a function of time, assuming that $\theta(t = 0) = 0$. Compare this plot with the Neato motion that you observe in the video. Is the estimate of θ accurate?