Lecture 21

Integration: Left, Right and Trapezoid Rules

The Left and Right endpoint rules

In this section, we wish to approximate a definite integral
\[ \int_a^b f(x) \, dx, \]
where \( f(x) \) is a continuous function. In calculus we learned that integrals are (signed) areas and can be approximated by sums of smaller areas, such as the areas of rectangles. We begin by choosing points \( \{x_i\} \) that subdivide \([a, b]\):
\[ a = x_0 < x_1 < \ldots < x_{n-1} < x_n = b. \]
The subintervals \([x_{i-1}, x_i]\) determine the width \( \Delta x_i \) of each of the approximating rectangles. For the height, we learned that we can choose any height of the function \( f(x^*_i) \) where \( x^*_i \in [x_{i-1}, x_i] \). The resulting approximation is
\[ \int_a^b f(x) \, dx \approx \sum_{i=1}^{n} f(x^*_i) \Delta x_i. \]

To use this to approximate integrals with actual numbers, we need to have a specific \( x^*_i \) in each interval. The two simplest (and worst) ways to choose \( x^*_i \) are as the left-hand point or the right-hand point of each interval. This gives concrete approximations which we denote by \( L_n \) and \( R_n \) given by
\[ L_n = \sum_{i=1}^{n} f(x_i-1) \Delta x_i \quad \text{and} \quad R_n = \sum_{i=1}^{n} f(x_i) \Delta x_i. \]

```matlab
function L = myleftsum(x,y)
% produces the left sum from data input.
% Inputs: x -- vector of the x coordinates of the partition
% y -- vector of the corresponding y coordinates
% Output: returns the approximate integral
n = length(x);
L = 0;
for i = 1:n-1
    % accumulate height times width
    L = L + y(i)*(x(i+1) - x(i));
end
end
```

82
Often we can take \( \{x_i\} \) to be *evenly spaced*, with each interval having the same width:

\[
h = \frac{b - a}{n},
\]

where \( n \) is the number of subintervals. If this is the case, then \( L_n \) and \( R_n \) simplify to

\[
L_n = h \sum_{i=0}^{n-1} f(x_i) \quad \text{and} \quad R_n = h \sum_{i=1}^{n} f(x_i).
\]

The foolishness of choosing left or right endpoints is illustrated in Figure 21.1. As you can see, for a very simple function like \( f(x) = 1 + .5x \), each rectangle of \( L_n \) is too short, while each rectangle of \( R_n \) is too tall. This will hold for any increasing function. For decreasing functions \( L_n \) will always be too large while \( R_n \) will always be too small.

**The Trapezoid rule**

Knowing that the errors of \( L_n \) and \( R_n \) are of opposite sign, a very reasonable way to get a better approximation is to take an average of the two. We will call the new approximation \( T_n \):

\[
T_n = \frac{L_n + R_n}{2}.
\]

This method also has a straightforward geometric interpretation. On each subrectangle we are using

\[
A_i = \frac{f(x_{i-1}) + f(x_i)}{2} \Delta x_i,
\]

which is exactly the area of the *trapezoid* with sides \( f(x_{i-1}) \) and \( f(x_i) \). We thus call the method the trapezoid method. See Figure 21.2. We can rewrite \( T_n \) as

![Figure 21.1: The left and right sums, \( L_n \) and \( R_n \).](image)
The trapezoid rule, \( T_n \), is given by:

\[
T_n = \sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2} \Delta x_i.
\]

In the evenly spaced case, we can write this as:

\[
T_n = \frac{h}{2} (f(x_0) + 2f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n)).
\]  

**Caution:** The convention used here is to begin numbering the points at 0, i.e. \( x_0 = a \); this allows \( n \) to be the number of subintervals and the index of the last point \( x_n \). However, MATLAB’s indexing convention begins at 1. Thus, when programming in MATLAB, the first entry in \( x \) will be \( x_0 \), i.e. \( x(1) = x_0 \) and \( x(n+1) = x_n \).

If we are given data about the function, rather than a formula for the function, often the data are not evenly spaced. The following function program could then be used.

```matlab
function T = mytrap(x,y)
    % Calculates the Trapezoid rule approximation of the integral from data
    % Inputs: x -- vector of the x coordinates of the partition
    %         y -- vector of the corresponding y coordinates
    % Output: returns the approximate integral
    n = length(x);
    T = 0;
    for i = 1:n-1
        % accumulate twice the signed area of the trapezoids
        T = T + (y(i) + y(i+1)) * (x(i+1) - x(i));
    end
    T = T/2; % correct for the missing 1/2
end
```
Using the Trapezoid rule for areas in the plane

In multi-variable calculus you were supposed to learn that you can calculate the area of a region $R$ in the plane by calculating the line integral

$$A = -\oint_C y \, dx,$$  \hspace{1cm} (21.4)

where $C$ is the counter-clockwise curve around the boundary of the region. We can represent such a curve by consecutive points on it, i.e. $\bar{x} = (x_0, x_1, x_2, \ldots, x_{n-1}, x_n)$, and $\bar{y} = (y_0, y_1, y_2, \ldots, y_{n-1}, y_n)$. Since we are assuming the curve ends where it begins, we require $(x_n, y_n) = (x_0, y_0)$. Applying the trapezoid method to the integral (21.4) gives

$$A = -\sum_{i=1}^{n} \frac{y_{i-1} + y_i}{2} (x_i - x_{i-1}).$$

This formula then is the basis for calculating areas when coordinates of boundary points are known, but not necessarily formulas for the boundaries such as in a land survey.

In the following script, we can use this method to approximate the area of a unit circle using $n$ points on the circle:

```matlab
% Calculates pi using a trapezoid approximation of the unit circle.
format long
n = 10; % evaluate points on the circle
t = linspace(0, 2*pi, n+1);
x = cos(t);
y = sin(t);
plot(x,y)
A = 0 % accumulate (twice) the trapezoid area
for i = 1:n
    A = A - (y(i)+y(i+1))*(x(i+1) - x(i));
end
A = A/2 % correct for the missing 1/2
```

Vector Operations using Slicing and Summing

In the programs above we used loops to explicitly accumulate sums. For example, in `mytrap` we had

```matlab
T = 0;
for i = 1:n-1
    T = T + .5*(y(i)+y(i+1))*(x(i+1) - x(i));
end
```

The alternative is to use vector operations by taking slices out of the vectors and using the `sum` function. We can replace the above code by

```matlab
T = .5*sum((y(1:n-1)+y(2:n)) .* (x(2:n)-x(1:n-1)));
```

Generally, explicit loops are easier to understand but vector operations are more efficient and compact.
Exercises

21.1 For the integral \( \int_{1}^{2} \sqrt{x} \, dx \) calculate \( L_4 \), \( R_4 \), and \( T_4 \) with even spacing (by hand, but use a calculator and a lot of digits) using formulas (21.1), (21.2) and (21.3). Find the percentage error of these approximations, using the exact value.

21.2 Write a well-commented MATLAB function program \texttt{myints} whose inputs are \( f \), \( a \), \( b \) and \( n \) and whose outputs are \( L \), \( R \) and \( T \), the left, right and trapezoid integral approximations for \( f \) on \([a,b]\) with \( n \) subintervals. To make it efficient,

- take advantage of the fact that \( \Delta x_i \) is constant,
- use \( x = \text{linspace}(a,b,n+1) \) to make the \( x \) values,
- use \( y = f(x) \) to make the \( y \) values,
- use \( \text{slice} \) and \( \text{sum} \) to add the 2nd to \( n \)th \( y \) entries once,
- and then add on the missing terms to obtain the left, right and trapezoid approximations.

Change to \texttt{format long} and apply your program to the integral \( \int_{1}^{2} \sqrt{x} \, dx \). Compare with the results of the previous exercise. Also find \( L_{100} \), \( R_{100} \) and \( T_{100} \) and the percentage errors of these approximations. Turn in the program and a brief summary of the results.