PART OF THE PICTURE: Numerical Methods

Mathematical models are used to solve problems in a wide variety of areas including science, engineering, business, and the social sciences. Many of these models consist of ordinary algebraic equations, differential equations, systems of equations, and so on, and the solution of the problem is obtained by finding solutions for these equations. Methods for solving such equations that can be implemented in a computer program are called **numerical methods,** and the development and analysis of such numerical methods is an important area of study in computer science.

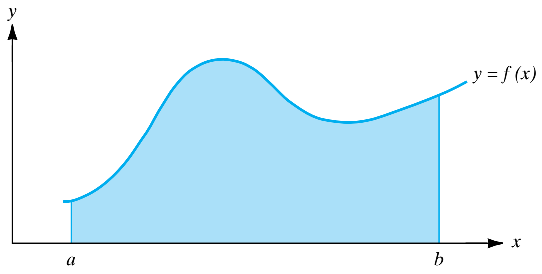
Some of the major types of problems in which numerical methods are routinely used include the following:

1. *Curve fitting*. In many applications, the solution of a problem requires analyzing data consisting of pairs of values to determine whether the items in these pairs are related. For example, a sociologist might wish to determine whether there is a linear relationship between educational level and income level.
2. *Solving equations*. Such problems deal with finding the value of a variable that satisfies a given equation.
3. *Integration*. The solution of many problems such as finding area under a curve, determining total revenue generated by sales of an item, calculating probabilities of certain events, and calculating work done by a force, require the evaluation of an integral. Often these integrals can only be evaluated using numerical techniques.
4. *Differential equations*. Differential equations are equations that involve one or more derivatives of unknown functions. Such equations play an important role in many applications, and several effective and efficient numerical methods for solving these equations have been developed.
5. *Solving linear systems*. Linear systems consist of several equations, each of which has several unknowns. A solution of such a system is a collection of values for these unknowns that satisfies all of the equations simultaneously.

Here we present a simple but practical introduction to one of these areas: integration. Examples from some of the other areas are described in the exercises and in later chapters.

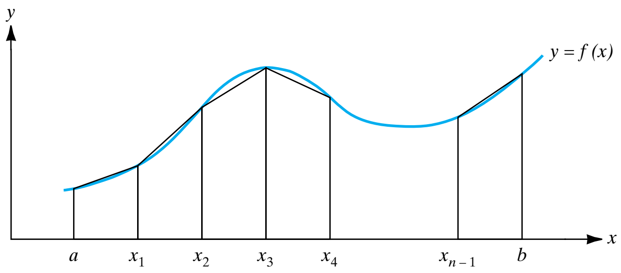
THE TRAPEZOID METHOD OF APPROXIMATING AREAS

One of the important problems in calculus is finding the area of a region having a curved boundary. Here we consider the simple case where the region is bounded below by the x-axis, above by the graph of a function $y = f(x)$, on the left by the vertical line $x = a$, and on the right by the vertical line $x = b$:



More generally, the problem is to approximate the integral $\int_a^b f(x)dx$.

One commonly used method for approximating this area is to divide the region into strips and approximate the area of each strip by using a trapezoid. More precisely, we cut the interval [*a*, *b*] into *n* parts, each of length $\Delta x = (b - n)/n$, using $n = 1$ equally spaced points $x_1, x_2, \ldots, x_{n-1}$. Locating the corresponding points on the curve and connecting consecutive points using line segments forms *n* trapezoids:



The sum of the areas of these trapezoids is approximately the area under the graph of *f*.

The bases of the first trapezoid are $y_0 = f(a)$ and $y_1 = f(x_1)$, and thus its area is[1]

$$\frac{1}{2}\Delta x(y_0 + y_1)$$

[1] The familiar formula from geometry for the area of a trapezoid with bases b_1 and b_2 and height *h* is $4h(b_1 + b_2)$. Note that in this problem, the bases are vertical and the "height" is horizontal.

Similarly, the area of the second trapezoid is

$$\frac{1}{2}\Delta x(y_1 + y_2)$$

where $y_2 = f(x_2)$, and so on. The sum of the areas of the n trapezoids is

$$\frac{1}{2}\Delta x(y_0 + y_1) + \frac{1}{2}\Delta x(y_1 + y_2) + \frac{1}{2}\Delta x(y_2 + y_2) + \quad + \frac{1}{2}\Delta x(y_{n-1} + y_n)$$

where $y_0, y_1, \ldots, y_{n-1}, y_n$ are the values of the function f at $a, x_1, \ldots, x_{n-1}, b$, respectively. Combining terms, we can write this sum more simply as

$$\Delta x\left(\frac{y_0 + y_n}{2} + y_1 + y_2 + \quad + y_n\right)$$

or, written more concisely using Σ (sigma) notation, as

$$\Delta x\left(\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i\right)$$

which is an approximation of the area under the curve.

This formula can be treated as an algorithm for the trapezoidal method, which we can use to define the following function. This particular definition reads the function values $y_0, y_1, \ldots, y_{n-1}, y_n$ from the keyboard.[2]

POP 8-1 The Trapezoidal Method (Definition).

```
/* trapezoidalArea computes the approximate area under a curve
 * for which a collection of y values at equally spaced x values
 * are entered from the keyboard.
 *
 *   Receive:      n, number of subintervals along the x axis
 *                 intLength, length of the interval on the x axis
 *   Precondition: The y values correspond to equally-spaced
 *                 x-values in the interval on the x axis.
 *   Return:       the approximate area under the curve
 ****************************************************************/

double trapezoidalArea(int n, double intLength)
{
   double yValue;
```

[2] `trapezoidalArea()` can be overloaded with other definitions that read the function values from a file, calculate them from an equation that defines the function, receive a `vector` of the function values from the caller (`vector` objects are discussed in Chapter 10), and so on.

```
   cout << "First y value? ";
   cin >> yValue;                         // the first y value
   double sum = yValue / 2.0;             // start sum at 1st term in
                                          //  the trap. formula
   for (int i = 1; i <= n - 1; i++)
   {
      cout >> "Next y value?  ";;
      cin >> yValue;                      // i-th y value
      sum += yValue;                      // add it to sum
   }

   cout << "Last y value?  ";
   cin >> yValue;                         // the last y value
   sum += yValue/ 2.0;                    // add 1/2 of it to sum

   double deltaX = intLength / double(n);

   return deltaX * sum;                   // total area of trapezoids
}
```

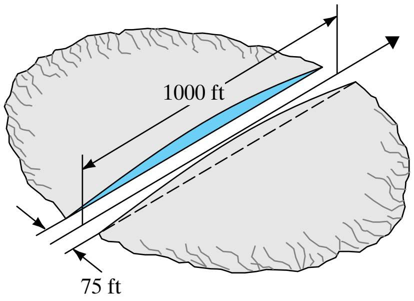
Since there are many different problems to which the trapezoidal method can be applied, it makes sense to store this function in a library (e.g., `MyMath`) for easy reuse. Of course, the prototype of the function must be stored in the library's header file. There are also many other methods for approximating areas such as the rectangle method, the midpoint method, Gaussian quadrature, and Simpson's rule (see the programming problems at the end of this chapter). One could develop an entire library of functions that implement these various methods.

There are many problems where it is necessary to compute the area under a curve (or the more general problem of calculating an integral). We will now consider one such real-world problem and solve it using the trapezoidal method.

PROBLEM: ROAD CONSTRUCTION

A construction company has contracted to build a highway for the state highway commission. Several sections of this highway must pass through hills from which large amounts of dirt must be excavated to provide a flat and level roadbed. For example, one section that is 1000 feet in length must pass through a hill whose height (in feet) above the roadbed has been measured at equally spaced distances and tabulated as follows:

Distance	Height
0	0
100	6
200	10
300	13
400	17
500	22
600	25
700	20
800	13
900	5
1000	0

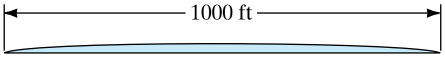


To estimate the construction costs, the company needs to know the volume of dirt that must be excavated from the hill.

OBJECT-CENTERED DESIGN

PRELIMINARY ANALYSIS. To estimate the volume of dirt to be removed, we will assume that the height of the hill does not vary from one side of the road to the other. The volume can then be calculated as

Volume = (cross-sectional area of the hill) × (width of the road)



The cross-sectional area of the hill can be computed using the trapezoidal method.

BEHAVIOR. The program should display on the screen a prompt for the length of the roadway passing through the hill and the number of points at which the height was measured and read those values. The program should then pass these values to `trapezoidalArea()` to compute the cross-sectional area of the hill. It should then display a prompt for the width of the roadway, read this value from the keyboard, and then compute and display the volume of dirt to be removed.

OBJECTS. From our behavioral description, we can identify the following objects in this problem:

	Software Objects		
Problem Objects	**Type**	**Kind**	**Name**
The length of roadway through the hill	varying	`double`	*roadLength*
The number of height measurements	varying	`int`	*numPoints*
The cross-sectional area of the hill	varying	`double`	*hillCrossSectionalArea*
The width of the road	varying	`double`	*roadWidth*
The volume of dirt	varying	`double`	*dirtVolume*

We can then specify the task of the program as follows:

Input: The length of a section of road
The number of height measurements at equally spaced points along this section
The collection of heights
The width of the road

Output: The volume of dirt to be removed

OPERATIONS. From our behavioral description, making a list of the operations is straightforward:

i. Read a double (*roadLength*) and an integer (*numPoints*) from the keyboard.
ii. Compute the cross-sectional area of the hill (`TrapezoidalArea()`).
iii. Initialize a real variable with a real value (*hillCrossSectionalArea*, *dirtVolume*).
iv. Multiply two real values.

ALGORITHM. These operations can be organized into an algorithm, as follows:

Algorithm for Highway Construction Problem

```
1. Prompt for and read the length of the roadway through the hill into roadLength.
2. Prompt for and read the number of height measurements into numPoints.
3. Compute the approximate cross-sectional area of the hill in hillCrossSectionalArea.
4. Prompt for and read the width of the road into roadWidth.
5. Compute the volume of dirt to be removed:
       $dirtVolume = hillCrossSectionalArea \times roadWidth$.
```

CODING AND TESTING. The following program implements this algorithm.

POP 8-2 Road Construction.

```
/* roadConstruct.cpp uses the trapezoidal method to find the volume
 *  of dirt that must be removed to construct a road through a hill.
 *
 *  Input:  the length of roadway through the hill,
 *          the width of the roadway
 *  Output: the approximate volume of dirt removed
 ******************************************************************/
```

```
#include <iostream>             // cin, cout. >>, <<
using namespace std;

double trapezoidalArea(int n, double totalLength);

int main()
{

   cout << "Enter the length of roadway through the hill: ";
   double roadLength;                           // get the road length
   cin >> roadLength;

   cout << "Enter the number of points at which hill elevation "
           "was measured: ";
   int numPoints;
   cin >> numPoints;

   cout << "Enter the hill elevations (y values) at the " << numPoints
        << " equally-spaced points.\n  The amount of dirt to be removed"
           " from the hill will be computed.\n\n";

                                               // compute X-sect. area
   double hillCrossSectionalArea
                           = trapezoidalArea(numPoints-1, roadLength);
   cout << "Enter the width of the roadway: ";
   double roadWidth;                           // get the road width
   cin >> roadWidth;
                                               // compute volume
   double dirtVolume = roadWidth * hillCrossSectionalArea;
                                               // display volume
   cout << "\n The volume of dirt to be removed is approximately "
        << dirtVolume << " cubic units." << endl;
}

/*** Insert definition of the function trapezoidalArea()
     from POP 8-1 here. ***/
```

Sample run:

```
Enter the length of roadway through the hill: 1000
Enter the number of points at which hill elevation was measured: 11
Enter the hill elevations (y values) at the 11 equally-spaced points.
The amount of dirt to be removed from the hill will be computed.

First y value? 0
Next y value?  6
Next y value?  10
Next y value?  13
Next y value?  17
Next y value?  22
Next y value?  25
Next y value?  20
Next y value?  13
Next y value?  5
Last y value?  0
Enter the width of the roadway: 75

The volume of dirt to be removed is approximately 982500 cubic units.
```

PROGRAMMING PROBLEMS

1. Another method of numerical integration that generally produces better approximations than the trapezoidal method is based on the use of parabolas and is known as *Simpson's rule*. In this method, the interval [*a, b*] is divided into an even number *n* of subintervals, each of length Δ*x*, and the sum

$$\frac{\Delta x}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \qquad + 2y_{n-2} + 4y_{n-1} + y_n)$$

is used to find the area under the graph of *f* over the interval [*a*, *b*]. Write a method `simpsonArea()` like `trapezoidalArea()` in POP 8-1 but use Simpson's rule. Then modify the program in POP 8-2 to use this method to find the volume of dirt removed.

2. The work done (in joules) by a force (in newtons) that is applied at an angle θ (radians) as it moves an object from *x* = *a* to *x* = *b* on the *x* axis (with meters as units) is given by

$$w = \cos\theta \int_a^b F(x)dx$$

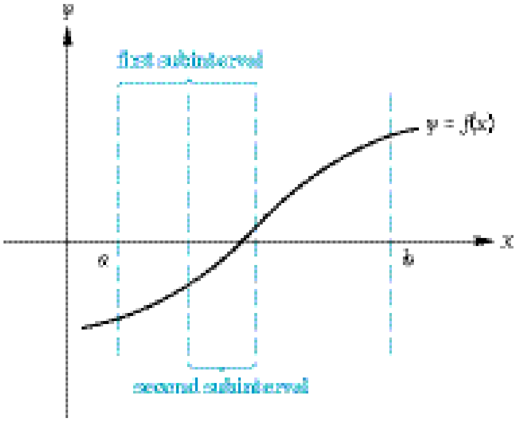
where *F*(*x*) is the force applied at point *x*. That is, the work done is the area under the graph of *y* = *F*(*x*) from *x* = *a* to *x* = *b* multiplied by cos θ. Write a program similar to that in POP 8-2 to compute the work done for θ = 0.35 radians, *a* = 10.0 m, *b* = 40.0 m, and the following forces measured at equally-spaced points from *a* to *b*: 0.0, 4.5, 9.0, 13.0, 14.0, 10.5, 12.0, 7.8, 5.0 (all in newtons).

3. Repeat Problem 2 but use Simpson's rule (see Problem 1).

4. Overload the method `trapezoidalArea()` in POP 8-1 with a definition that has as parameters the endpoints `a` and `b` of the interval and the number `n` of points of subdivision, and which calls some method `f()` to calculate the function values rather than input them. Test your method with a driver program and method `f()` that calculates x^2. (Note: The area under the graph of $y = x^2$ from *x* = *a* to *x* = *b* is $(b^3 - a^3) / 3$.)

5. Another area of numerical methods is equation solving. One method for finding an approximate solution of an equation *f*(*x*) = 0 for some function *f* is the *bisection method*. In this method, we begin with two numbers *a* and *b*, where the function values *f*(*a*) and *f*(*b*) have opposite signs. If *f* is continuous between *x* = *a* and *x* = *b*—that is, if there is no break in the graph of *y* = *f*(*x*) between these two values—then the graph of *f* must cross the *x*-axis at least once between *x* = *a* and *x* = *b*; thus, there must

be at least one solution of the equation $f(x) = 0$ between a and b. To locate one of these solutions, we first bisect the interval $[a, b]$ and determine in which half f changes sign, thereby locating a smaller subinterval containing a solution of the equation. We bisect this subinterval and determine in which half f changes sign; this gives a still smaller subinterval containing a solution.



Repeating this process gives a sequence of subintervals, each of which contains a solution of the equation and whose length is one-half that of the preceding interval. Write a method to implement the bisection method and use it in a program to find a solution of the equation $x^3 + x - 5 = 0$.