\[ q_1 = \int_0^1 f(x) \, dx = 0.099833416647 \quad \text{and} \quad q_2 = \int_0^1 P_3(x) \, dx = 0.099833333333, \]

with error \( 0.83314 \times 10^{-7} = 8.3314 \times 10^{-8} \)

Parts (a) and (b) of the example show how two techniques can produce the same approximation but have differing accuracy assurances. Remember that determining approximations is only part of our objective. An equally important part is to determine a bound for the error of the approximation.

**EXERCISE SET 1.1**

1. Show that the following equations have at least one solution in the given intervals
   a. \( x \cos x - 2x^2 + 3x - 1 = 0 \), \([0, 2, 0]\) and \([1, 2, 1, 3]\)
   b. \( (x - 2)^2 - \ln x = 0 \), \([1, 2]\) and \([e, 4]\)
   c. \( 2x \cos(2x) - (x - 2)^2 = 0 \), \([2, 3]\) and \([3, 4]\)
   d. \( x - (\ln x)^e = 0 \), \([4, 5]\)

2. Find intervals containing solutions to the following equations
   a. \( x - 3^x = 0 \)
   b. \( 4x^2 - e^x = 0 \)
   c. \( x^3 - 2x^2 - 4x + 2 = 0 \)
   d. \( x^3 + 4 \cos x + 1101 = 0 \)

3. Show that \( f'(x) = 0 \) at least once in the given intervals.
   a. \( f(x) = 1 - e^x + (e - 1) \sin((\pi/2)x) \), \([0, 1]\)
   b. \( f(x) = (x - 1) \tan x + x \sin \pi x \), \([0, 1]\)
   c. \( f(x) = x \sin \pi x - (x - 2) \ln x \), \([1, 2]\)
   d. \( f(x) = (x - 2) \sin x \ln(x + 2) \), \([-1, 3]\)

4. Find \( \max_{x \in [a, b]} |f(x)| \) for the following functions and intervals
   a. \( f(x) = (2 - e^x + 2x)/3 \), \([0, 1]\)
   b. \( f(x) = (4x - 3)/(x^2 - 2x) \), \([0.5, 1]\)
   c. \( f(x) = 2x \cos(2x) - (x - 2)^2 \), \([2, 4]\)
   d. \( f(x) = 1 + e^{-\cos(x)} \), \([1, 2]\)

5. Use the Intermediate Value Theorem and Rolle's Theorem to show that the graph of \( f(x) = x^3 + 2x + k \) crosses the x-axis exactly once, regardless of the value of the constant \( k \).

6. Suppose \( f \in C[a, b] \) and \( f'(x) \) exists on \( (a, b) \) Show that if \( f'(x) \neq 0 \) for all \( x \) in \( (a, b) \), then there can exist at most one number \( p \) in \( [a, b] \) with \( f(p) = 0 \)

7. Let \( f(x) = x^3 \)
   a. Find the second Taylor polynomial \( P_2(x) \) about \( x_0 = 0 \)
   b. Find \( P_2(0.5) \) and the actual error in using \( P_2(0.5) \) to approximate \( f(0.5) \).
   c. Repeat part (a) using \( x_0 = 1 \).
   d. Repeat part (b) using the polynomial from part (c)

8. Find the third Taylor polynomial \( P_3(x) \) for the function \( f(x) = \sqrt{x + 1} \) about \( x_0 = 0 \). Approximate \( \sqrt{0.5}, \sqrt{0.75}, \sqrt{1/2}, \) and \( \sqrt{1.5} \) using \( P_3(x) \), and find the actual errors.

9. Find the second Taylor polynomial \( P_2(x) \) for the function \( f(x) = e^x \cos x \) about \( x_0 = 0 \).
   a. Use \( P_2(0.5) \) to approximate \( f(0.5) \). Find an upper bound for error \(|f(0.5) - P_2(0.5)|\) using the error formula, and compare it to the actual error.
1.2 Round-off Errors and Computer Arithmetic

The arithmetic performed by a calculator or computer is different from the arithmetic in our algebra and calculus courses. From your past experience, you might expect that we always have as true statements such things as $2 + 2 = 4$, $4 \cdot 8 = 32$, and $(\sqrt{3})^2 = 3$. In standard computational arithmetic, we expect exact results for $2 + 2 = 4$ and $4 \cdot 8 = 32$, but we will not have precisely $(\sqrt{3})^2 = 3$. To understand why this is true, we must explore the world of finite-digit arithmetic.

In our traditional mathematical world, we permit numbers with an infinite number of digits. The arithmetic we use in this world defines $\sqrt{3}$ as that unique positive number that when multiplied by itself produces the integer 3. In the computational world, however, each representable number has only a fixed and finite number of digits. This means, for example, that only rational numbers—and not even all of these—can be represented exactly. Since $\sqrt{3}$ is not rational, it is given an approximate representation, one whose square will not be precisely 3, although it will likely be sufficiently close to 3 to be acceptable in most situations. In most cases, then, this machine arithmetic is satisfactory and passes without notice or concern, but at times problems arise because of this discrepancy.

The error that is produced when a calculator or computer is used to perform real-number calculations is called round-off error. It occurs because the arithmetic performed in a machine involves numbers with only a finite number of digits, with the result that calculations are performed with only approximate representations of the actual numbers. In a typical computer, only a relatively small subset of the real number system is used for the representation of all the real numbers. This subset contains only rational numbers, both positive and negative, and stores the fractional part together with an exponential part.

In 1985, the IEEE (Institute of Electrical and Electronics Engineers) published a report called *Binary Floating Point Arithmetic Standard 754–1985*. Formats were specified for single, double, and extended precisions, and these standards are generally followed by all microcomputer manufacturers using floating-point hardware. For example, the numerical coprocessor for PCs implements a 64-bit (binary digit) representation for a real number, called a long real. The first bit is a sign indicator, denoted $s$. This is followed by an 11-bit exponent, $e$, called the characteristic, and a 52-bit binary fraction, $f$, called the mantissa. The base for the exponent is 2.

Since 52 binary digits correspond to between 15 and 16 decimal digits, we can assume that a number represented in this system has at least 15 decimal digits of precision. The exponent of 11 binary digits gives a range of 0 to $2^{11} - 1 = 2047$. However, using only positive integers for the exponent would not permit an adequate representation of numbers with small magnitude. To ensure that numbers with small magnitude are equally representable, 1023 is subtracted from the characteristic, so the range of the exponent is actually from $-1023$ to 1024.

To save storage and provide a unique representation for each floating-point number, a normalization is imposed. Using this system gives a floating-point number of the form

$$(-1)^e 2^{e-1023} (1 + f)$$

Consider, for example, the machine number

$$0 10000000011 1011001000100000000000000000000000000000000000000000000000000000000.$$

Expect error due to rounding whenever computations are performed using numbers that are not powers of 2. Keeping this error under control is extremely important when the number of calculations is large.
The leftmost bit is zero, which indicates that the number is positive. The next 11 bits, 10000000001, giving the characteristic, are equivalent to the decimal number
\[ c = 1 \cdot 2^{10} + 0 \cdot 2^9 + \cdots + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1024 + 2 + 1 = 1027. \]
The exponential part of the number is therefore \( 2^{1027-1023} = 2^4 \). The final 52 bits specify that the mantissa is
\[ f = 1 \cdot \left( \frac{1}{2} \right)^1 + 1 \cdot \left( \frac{1}{2} \right)^3 + 1 \cdot \left( \frac{1}{2} \right)^4 + 1 \cdot \left( \frac{1}{2} \right)^5 + 1 \cdot \left( \frac{1}{2} \right)^8 + 1 \cdot \left( \frac{1}{2} \right)^{12}. \]
As a consequence, this machine number precisely represents the decimal number
\[
(-1)^4 2^{1023} (1 + f) = (-1)^0 \cdot 2^{1027-1023} \left( 1 + \left( \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096} \right) \right) = 27.56640625.
\]
However, the next smallest machine number is
\[ 0 100000000011 1011100100001111111111111111111111111111111111, \]
and the next largest machine number is
\[ 0 100000000011 10111001000010000000000000000000000000000001. \]
This means that our original machine number represents not only 27.56640625, but also half of the real numbers that are between 27.56640625 and its two nearest machine-number neighbors. To be precise, it represents any real number in the interval
\[
[27.56640624999998223643160599749533221893310546875, 27.56640625000000017763568394002504646778106689453125].
\]
The smallest normalized positive number that can be represented has \( s = 0, c = 1, \) and \( f = 0 \) and is equivalent to
\[ 2^{-1022} \quad (1 + 0) \approx 0.2225 \times 10^{-307}, \]
and the largest has \( s = 0, c = 2046, \) and \( f = 1 - 2^{-52} \) and is equivalent to
\[ 2^{1023} \cdot (2 - 2^{-52}) \approx 0.17977 \times 10^{109}. \]
Numbers occurring in calculations that have a magnitude less than
\[ 2^{-1022} \quad (1 + 0) \]
result in \textit{underflow} and are generally set to zero. Numbers greater than
\[ 2^{1023} \quad (2 - 2^{-52}) \]
result in \textit{overflow} and typically cause the computations to stop (unless the program has been designed to detect this occurrence). Note that there are two representations for the number zero; a positive 0 when \( s = 0, c = 0, \) and \( f = 0, \) and a negative 0 when \( s = 1, c = 0, \) and \( f = 0. \)
The use of binary digits tends to conceal the computational difficulties that occur when a finite collection of machine numbers is used to represent all the real numbers. To examine these problems, we now assume for simplicity that machine numbers are represented in the normalized decimal floating-point form

$$\pm 0.d_1d_2 \ldots d_k \times 10^n, \quad 1 \leq d_i \leq 9, \quad \text{and} \quad 0 \leq d_i \leq 9,$$

for each $i = 2, \ldots, k$. Numbers of this form are called $k$-digit decimal machine numbers.

Any positive real number within the numerical range of the machine can be normalized to the form

$$y = 0.d_1d_2 \ldots d_k d_{k+1}d_{k+2} \ldots \times 10^n$$

The floating-point form of $y$, denoted $fl(y)$, is obtained by terminating the mantissa of $y$ at $k$ decimal digits. There are two ways of performing this termination. One method, called chopping, is to simply chop off the digits $d_{k+1}d_{k+2} \ldots$. This produces the floating-point form

$$fl(y) = 0.d_1d_2 \ldots d_k \times 10^n.$$  

The other method, called rounding, adds $5 \times 10^{n-(k+1)}$ to $y$ and then chops the result to obtain a number of the form

$$fl(y) = 0.d_1d_2 \ldots \delta_k \times 10^n.$$  

So, when rounding, if $d_{k+1} \geq 5$, we add 1 to $d_k$ to obtain $fl(y)$; that is, we round up. When $d_{k+1} < 5$, we merely chop off all but the first $k$ digits; so we round down. If we round down, then $\delta_i = d_i$, for each $i = 1, 2, \ldots, k$. However, if we round up, the digits (and even the exponent) might change.

**Example 1**

The number $\pi$ has an infinite decimal expansion of the form $\pi = 3.14159265 \ldots$. Written in normalized decimal form, we have

$$\pi = 0.314159265 \ldots \times 10^1.$$  

The floating-point form of $\pi$ using five-digit chopping is

$$fl(\pi) = 0.31415 \times 10^1 = 3.1415.$$  

Since the sixth digit of the decimal expansion of $\pi$ is a 9, the floating-point form of $\pi$ using five-digit rounding is

$$fl(\pi) = (0.31415 + 0.00001) \times 10^1 = 3.1416.$$  

The following definition describes two methods for measuring approximation errors.

**Definition 1.15**

If $p^*$ is an approximation to $p$, the absolute error is $|p - p^*|$, and the relative error is $|p - p^*|/|p|$, provided that $p \neq 0$.

Consider the absolute and relative errors in representing $p$ by $p^*$ in the following example.
**Example 2**

We often cannot find an accurate value for the true error in an approximation. Instead we find a bound for the error, which gives us a "worst-case" error.

(a) If \( p = 0.3000 \times 10^1 \) and \( p^* = 0.3100 \times 10^1 \), the absolute error is 0.1, and the relative error is \( 0.3333 \times 10^{-1} \).

(b) If \( p = 0.3000 \times 10^{-3} \) and \( p^* = 0.3100 \times 10^{-3} \), the absolute error is \( 0.1 \times 10^{-4} \), and the relative error is \( 0.3333 \times 10^{-1} \).

(c) If \( p = 0.3000 \times 10^4 \) and \( p^* = 0.3100 \times 10^4 \), the absolute error is \( 0.1 \times 10^3 \), and the relative error is again \( 0.3333 \times 10^{-1} \).

This example shows that the same relative error, \( 0.3333 \times 10^{-1} \), occurs for widely varying absolute errors. As a measure of accuracy, the absolute error can be misleading and the relative error more meaningful since the relative error takes into consideration the size of the value.

The following definition uses relative error to give a measure of significant digits of accuracy for an approximation.

**Definition 1.16**

The term significant digits is often used to loosely describe the number of decimal digits that appear to be accurate. This definition is more precise, and provides a continuous concept.

The number \( p^* \) is said to approximate \( p \) to \( r \) **significant digits** (or figures) if \( r \) is the largest nonnegative integer for which

\[
\frac{|p - p^*|}{|p|} \leq 5 \times 10^{-r}.
\]

Table 1.1 illustrates the continuous nature of significant digits by listing, for the various values of \( p \), the least upper bound of \( |p - p^*| \), denoted \( |p - p^*| \), when \( p^* \) agrees with \( p \) to four significant digits.

<table>
<thead>
<tr>
<th>( p )</th>
<th>0.1</th>
<th>0.5</th>
<th>100</th>
<th>1000</th>
<th>5000</th>
<th>9990</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \max</td>
<td>p - p^*</td>
<td>)</td>
<td>0.00005</td>
<td>0.00025</td>
<td>0.05</td>
<td>0.5</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Returning to the machine representation of numbers, we see that the floating-point representation \( fI(y) \) for the number \( y \) has the relative error

\[
\left| \frac{y - fI(y)}{y} \right|.
\]

If \( k \) decimal digits and chopping are used for the machine representation of

\[ y = d_1d_2 \ldots d_kd_{k+1} \ldots \times 10^n, \]

then

\[
\left| \frac{y - fI(y)}{y} \right| = \left| \frac{0.\,d_1d_2 \ldots d_kd_{k+1} \ldots \times 10^n - 0.\,d_1d_2 \ldots d_k \times 10^n}{0.\,d_1d_2 \ldots \times 10^n} \right| = \left| \frac{0.\,d_{k+1}d_{k+2} \ldots \times 10^{n-k}}{0.\,d_1d_2 \ldots \times 10^n} \right| = \left| \frac{0.\,d_{k+1}d_{k+2} \ldots}{0.\,d_1d_2 \ldots} \right| \times 10^{-k}.
\]
Since \( d_1 \neq 0 \), the minimal value of the denominator is 0 1. The numerator is bounded above by 1. As a consequence,

\[
\left| \frac{y - f_l(y)}{y} \right| \leq \frac{1}{0 1} \times 10^{-k} = 10^{-k+1}.
\]

In a similar manner, a bound for the relative error when using \( k \)-digit rounding arithmetic is \( 0.5 \times 10^{-k+1} \). (See Exercise 24.)

Note that the bounds for the relative error using \( k \)-digit arithmetic are independent of the number being represented. This result is due to the manner in which the machine numbers are distributed along the real line. Because of the exponential form of the characteristic, the same number of decimal machine numbers is used to represent each of the intervals \([0.1, 1], [1, 10], \) and \([10, 100]\). In fact, within the limits of the machine, the number of decimal machine numbers in \([10^n, 10^{n+1}]\) is constant for all integers \( n \).

In addition to inaccurate representation of numbers, the arithmetic performed in a computer is not exact. The arithmetic involves manipulating binary digits by various shifting, or logical, operations. Since the actual mechanics of these operations are not pertinent to this presentation, we shall devise our own approximation to computer arithmetic. Although our arithmetic will not give the exact picture, it suffices to explain the problems that occur.

Assume that the floating-point representations \( f_l(x) \) and \( f_l(y) \) are given for the real numbers \( x \) and \( y \) and that the symbols \( \oplus, \ominus, \otimes, \oslash \) represent machine addition, subtraction, multiplication, and division operations, respectively. We will assume a finite-digit arithmetic given by

\[
\begin{align*}
x \oplus y &= f_l(f_l(x) + f_l(y)), & x \ominus y &= f_l(f_l(x) \times f_l(y)), \\
x \otimes y &= f_l(f_l(x) - f_l(y)), & x \oslash y &= f_l(f_l(x) \div f_l(y))
\end{align*}
\]

This arithmetic corresponds to performing exact arithmetic on the floating-point representations of \( x \) and \( y \) and then converting the exact result to its finite-digit floating-point representation.

Rounding arithmetic is easily implemented in a CAS. The Maple command

\[>\text{Digits:=t;}\]

causes all arithmetic to be rounded to \( t \) digits. For example, \( f_l(f_l(x) + f_l(y)) \) is performed using \( t \)-digit rounding arithmetic by

\[>\text{evalf(evalf(x) + evalf(y))};\]

Implementing \( t \)-digit chopping arithmetic is more difficult and requires a sequence of steps or a procedure. Exercise 27 explores this problem.

**Example 3**

Suppose that \( x = \frac{5}{7}, y = \frac{3}{4} \), and that five-digit chopping is used for arithmetic calculations involving \( x \) and \( y \). Table 1.2 lists the values of these computer-type operations on \( f_l(x) = 0.71428 \times 10^6 \) and \( f_l(y) = 0.33333 \times 10^0 \).

Since the maximum relative error for the operations in Example 3 is \( 0.267 \times 10^{-3} \), the arithmetic produces satisfactory five-digit results. Suppose, however, that we also have \( u = 0.714251, v = 98765.9, \) and \( w = 0.111111 \times 10^{-4} \), so that \( f_l(u) = 0.71425 \times 10^6, \)

\[>\text{Digits:=t;}\]

causes all arithmetic to be rounded to \( t \) digits. For example, \( f_l(f_l(x) + f_l(y)) \) is performed using \( t \)-digit rounding arithmetic by

\[>\text{evalf(evalf(x) + evalf(y))};\]

Implementing \( t \)-digit chopping arithmetic is more difficult and requires a sequence of steps or a procedure. Exercise 27 explores this problem.

**Example 3**

Suppose that \( x = \frac{5}{7}, y = \frac{3}{4} \), and that five-digit chopping is used for arithmetic calculations involving \( x \) and \( y \). Table 1.2 lists the values of these computer-type operations on \( f_l(x) = 0.71428 \times 10^6 \) and \( f_l(y) = 0.33333 \times 10^0 \).

Since the maximum relative error for the operations in Example 3 is \( 0.267 \times 10^{-3} \), the arithmetic produces satisfactory five-digit results. Suppose, however, that we also have \( u = 0.714251, v = 98765.9, \) and \( w = 0.111111 \times 10^{-4} \), so that \( f_l(u) = 0.71425 \times 10^6, \)
Table 1.2

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Actual value</th>
<th>Absolute error</th>
<th>Relative error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \oplus y$</td>
<td>$0.10476 \times 10^1$</td>
<td>$22/21$</td>
<td>$0.190 \times 10^{-4}$</td>
<td>$0.182 \times 10^{-4}$</td>
</tr>
<tr>
<td>$x \ominus y$</td>
<td>$0.38095 \times 10^0$</td>
<td>$8/21$</td>
<td>$0.238 \times 10^{-5}$</td>
<td>$0.625 \times 10^{-5}$</td>
</tr>
<tr>
<td>$x \otimes y$</td>
<td>$0.23809 \times 10^0$</td>
<td>$5/21$</td>
<td>$0.524 \times 10^{-5}$</td>
<td>$0.220 \times 10^{-4}$</td>
</tr>
<tr>
<td>$x \div y$</td>
<td>$0.21428 \times 10^1$</td>
<td>$15/7$</td>
<td>$0.571 \times 10^{-4}$</td>
<td>$0.267 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

$fI(v) = 0.98765 \times 10^5$, and $fI(w) = 0.11111 \times 10^{-4}$. (These numbers were chosen to illustrate some problems that can arise with finite-digit arithmetic.)

In Table 1.3, $x \ominus u$ results in a small absolute error but a large relative error. The subsequent division by the small number $w$ or multiplication by the large number $v$ magnifies the absolute error without modifying the relative error. The addition of the large and small numbers $u$ and $v$ produces large absolute error but not large relative error.

Table 1.3

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Actual value</th>
<th>Absolute error</th>
<th>Relative error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \ominus u$</td>
<td>$0.30000 \times 10^{-4}$</td>
<td>$0.34714 \times 10^{-4}$</td>
<td>$0.471 \times 10^{-5}$</td>
<td>$0.136$</td>
</tr>
<tr>
<td>$(x \ominus u) \otimes w$</td>
<td>$0.27000 \times 10^1$</td>
<td>$0.31243 \times 10^1$</td>
<td>$0.424$</td>
<td>$0.136$</td>
</tr>
<tr>
<td>$(x \ominus u) \otimes v$</td>
<td>$0.29629 \times 10^1$</td>
<td>$0.34285 \times 10^1$</td>
<td>$0.465$</td>
<td>$0.136$</td>
</tr>
<tr>
<td>$u \otimes v$</td>
<td>$0.98765 \times 10^5$</td>
<td>$0.98766 \times 10^5$</td>
<td>$0.161 \times 10^1$</td>
<td>$0.163 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

One of the most common error-producing calculations involves the cancelation of significant digits due to the subtraction of nearly equal numbers. Suppose two nearly equal numbers $x$ and $y$, with $x > y$, have the $k$-digit representations

$$fI(x) = 0.d_1d_2 \ldots d_p\alpha_{p+1}\alpha_{p+2} \ldots \alpha_k \times 10^p,$$

and

$$fI(y) = 0.d_1d_2 \ldots d_p\beta_{p+1}\beta_{p+2} \ldots \beta_k \times 10^p.$$

The floating-point form of $x - y$ is

$$fI(fI(x) - fI(y)) = 0.\sigma_{p+1}\sigma_{p+2} \ldots \sigma_k \times 10^{p-p},$$

where

$$0.\sigma_{p+1}\sigma_{p+2} \ldots \sigma_k = 0.\alpha_{p+1}\alpha_{p+2} \ldots \alpha_k - 0.\beta_{p+1}\beta_{p+2} \ldots \beta_k.$$

The floating-point number used to represent $x - y$ has at most $k - p$ digits of significance. However, in most calculation devices, $x - y$ will be assigned $k$ digits, with the last $p$ being either zero or randomly assigned. Any further calculations involving $x - y$ retain the problem of having only $k - p$ digits of significance, since a chain of calculations is no more accurate than its weakest portion.

If a finite-digit representation or calculation introduces an error, further enlargement of the error occurs when dividing by a number with small magnitude (or, equivalently, when multiplying by a number with large magnitude). Suppose, for example, that the number $z$ has the finite-digit approximation $z + \delta$, where the error $\delta$ is introduced by representation...
or by previous calculation if we now divide by \( e = 10^{-n} \), where \( n > 0 \), then

\[
\frac{z}{e} \approx f l \left( \frac{f l(z)}{f l(e)} \right) = (z + \delta) \times 10^n
\]

Thus, the absolute error in this approximation, \( |\delta| \times 10^n \), is the original absolute error, \( |\delta| \), multiplied by the factor \( 10^n \).

**Example 4**

Let \( p = 0.54617 \) and \( q = 0.54601 \). The exact value of \( r = p - q \) is \( r = 0.00016 \). Suppose the subtraction is performed using four-digit arithmetic. Rounding \( p \) and \( q \) to four digits gives \( p^* = 0.5462 \) and \( q^* = 0.5460 \), respectively, and \( r^* = p^* - q^* = 0.0002 \) is the four-digit approximation to \( r \). Since

\[
\frac{|r - r^*|}{|r|} = \frac{|0.00016 - 0.0002|}{0.00016} = 0.25,
\]

the result has only one significant digit, whereas \( p^* \) and \( q^* \) were accurate to four and five significant digits, respectively.

If chopping is used to obtain the four digits, the four-digit approximations to \( p, q, \) and \( r \) are \( p^* = 0.5461, q^* = 0.5460, \) and \( r^* = p^* - q^* = 0.0001 \). This gives

\[
\frac{|r - r^*|}{|r|} = \frac{|0.00016 - 0.0001|}{0.00016} = 0.375,
\]

which also results in only one significant digit of accuracy.

The loss of accuracy due to round-off error can often be avoided by a reformulation of the problem, as illustrated in the next example.

**Example 5**

The quadratic formula states that the roots of \( ax^2 + bx + c = 0 \), when \( a \neq 0 \), are

\[
x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.
\]

Using four-digit rounding arithmetic, consider this formula applied to the equation \( x^2 + 62.10x + 1 = 0 \), whose roots are approximately

\[
x_1 = -0.01610723 \quad \text{and} \quad x_2 = -62.08390.
\]

In this equation, \( b^2 \) is much larger than \( 4ac \), so the numerator in the calculation for \( x_1 \) involves the subtraction of nearly equal numbers. Since

\[
\sqrt{b^2 - 4ac} = \sqrt{(62.10)^2 - (4.000)(1.000)(1.000)}
\]

\[
= \sqrt{3856} - 4.000 = \sqrt{3852} = 62.06,
\]

we have

\[
fl(x_1) = \frac{-62.10 + 62.06}{2.000} = \frac{-0.04000}{2.000} = -0.02000,
\]
a poor approximation to \( x_1 = -0.01611 \), with the large relative error
\[
\frac{|-0.01611 + 0.02000|}{|-0.01611|} \approx 2.4 \times 10^{-1}
\]

On the other hand, the calculation for \( x_2 \) involves the addition of the nearly equal numbers \(-b\) and \(-\sqrt{b^2 - 4ac}\). This presents no problem since
\[
fl(x_2) = \frac{-62.10 - 62.06}{2.000} = \frac{-124.2}{2.000} = -62.10
\]
has the small relative error
\[
\frac{|-62.08 + 62.10|}{|-62.08|} \approx 3.2 \times 10^{-4}.
\]

To obtain a more accurate four-digit rounding approximation for \( x_1 \), we change the form of the quadratic formula by rationalizing the numerator:
\[
x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \cdot \frac{\left( -b - \sqrt{b^2 - 4ac} \right)}{\left( -b - \sqrt{b^2 - 4ac} \right)} = \frac{b^2 - (b^2 - 4ac)}{2a(-b - \sqrt{b^2 - 4ac})},
\]
which simplifies to an alternate quadratic formula
\[
x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}}.
\]
(1.2)

Using (1.2) gives
\[
fl(x_1) = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.01610,
\]
which has the small relative error \( 6.2 \times 10^{-4} \).

The rationalization technique can also be applied to give the following alternative quadratic formula for \( x_2 \):
\[
x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}}.
\]
(1.3)

This is the form to use if \( b \) is a negative number. In Example 5, however, the mistaken use of this formula for \( x_2 \) would result in not only the subtraction of nearly equal numbers, but also the division by the small result of this subtraction. The inaccuracy that this combination produces,
\[
fl(x_2) = \frac{-2c}{b - \sqrt{b^2 - 4ac}} = \frac{-2.000}{62.10 - 62.06} = \frac{-2.000}{0.04000} = -50.00,
\]
has the large relative error \( 1.9 \times 10^{-1} \).

Accuracy loss due to round-off error can also be reduced by rearranging calculations, as shown in the next example.
Example 6 Evaluate \( f(x) = x^3 - 6.1x^2 + 3.2x + 1.5 \) at \( x = 4.71 \) using three-digit arithmetic.

Table 1.4 gives the intermediate results in the calculations. Verify these results carefully to be sure that your notion of finite-digit arithmetic is correct. Note that the three-digit chopping values simply retain the leading three digits, with no rounding involved, and differ significantly from the three-digit rounding values.

<table>
<thead>
<tr>
<th></th>
<th>( x )</th>
<th>( x^2 )</th>
<th>( x^3 )</th>
<th>( 6.1x^2 )</th>
<th>( 3.2x )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exact</strong></td>
<td>4.71</td>
<td>22.1841</td>
<td>104.487111</td>
<td>135 32301</td>
<td>15 072</td>
</tr>
<tr>
<td><strong>Three-digit (chopping)</strong></td>
<td>4.71</td>
<td>22.1</td>
<td>104.</td>
<td>134</td>
<td>15.0</td>
</tr>
<tr>
<td><strong>Three-digit (rounding)</strong></td>
<td>4.71</td>
<td>22.2</td>
<td>105.</td>
<td>135</td>
<td>15.1</td>
</tr>
</tbody>
</table>

To illustrate the situation, let us look at the calculations involved with finding \( x^3 \) using three-digit rounding arithmetic. First we find

\[ x^2 = 4.71^2 = 22.1841 \quad \text{and round this to} \quad 22.2 \]

Then we use this value of \( x^2 \) to find

\[ x^3 = x^2 \cdot x = 22.2 \cdot 4.71 = 104.562 \quad \text{and round this to} \quad 105. \]

**Exact:**

\[ f(4.71) = 104.487111 - 135.32301 + 15.072 + 1.5 \]

\[ = -14.263899; \]

**Three-digit (chopping):**

\[ f(4.71) = ((104. - 134.) + 15.0) + 1.5 = -13.5; \]

**Three-digit (rounding):**

\[ f(4.71) = ((105. - 135.) + 15.1) + 1.5 = -13.4; \]

The relative errors for the three-digit methods are

\[ \left| \frac{-14.263899 + 13.5}{-14.263899} \right| \approx 0.05, \quad \text{for chopping} \]

and

\[ \left| \frac{-14.263899 + 13.4}{-14.263899} \right| \approx 0.06, \quad \text{for rounding}. \]

As an alternative approach, \( f(x) \) can be written in a nested manner as

\[ f(x) = x^3 - 6.1x^2 + 3.2x + 1.5 = ((x - 6.1)x + 3.2)x + 1.5. \]

This gives

**Three-digit (chopping):**

\[ f(4.71) = ((4.71 - 6.1)4.71 + 3.2)4.71 + 1.5 = -14.2; \]

and a three-digit rounding answer of \(-14.3\). The new relative errors are

**Three-digit (chopping):**

\[ \left| \frac{-14.263899 + 14.2}{-14.263899} \right| \approx 0.0045; \]

**Three-digit (rounding):**

\[ \left| \frac{-14.263899 + 14.3}{-14.263899} \right| \approx 0.0025. \]
Nesting has reduced the relative error for the chopping approximation to less than 10% of that obtained initially. For the rounding approximation, the improvement has been even more dramatic; the error in this case has been reduced by more than 95%.

Polynomials should always be expressed in nested form before performing an evaluation, because this form minimizes the number of arithmetic calculations. The decreased error in Example 6 is due to the reduction in computations from four multiplications and three additions to two multiplications and three additions. One way to reduce round-off error is to reduce the number of error-producing computations.

**EXERCISE SET 1.2**

1. Compute the absolute error and relative error in approximations of \( p \) by \( p^* \).
   
   a. \( p = \pi, p^* = 22/7 \)  
   b. \( p = \pi, p^* = 3.1416 \)
   c. \( p = e, p^* = 2.718 \)  
   d. \( p = \sqrt{2}, p^* = 1.414 \)
   e. \( p = e^6, p^* = 22000 \)  
   f. \( p = 10^9, p^* = 1400 \)
   g. \( p = 91, p^* = 39900 \)  
   h. \( p = 91, p^* = \sqrt{18\pi}(9/e)^9 \)

2. Find the largest interval in which \( p^* \) must lie to approximate \( p \) with relative error at most \( 10^{-4} \) for each value of \( p \).
   
   a. \( \pi \)  
   b. \( e \)  
   c. \( \sqrt{2} \)  
   d. \( \sqrt{7} \)

3. Suppose \( p^* \) must approximate \( p \) with relative error at most \( 10^{-3} \). Find the largest interval in which \( p^* \) must lie for each value of \( p \).
   
   a. 150  
   b. 900  
   c. 1500  
   d. 90

4. Perform the following computations (i) exactly, (ii) using three-digit chopping arithmetic, and (iii) using three-digit rounding arithmetic. (iv) Compute the relative errors in parts (ii) and (iii).
   
   a. \( \frac{\pi}{3} + \frac{1}{3} \)  
   b. \( \frac{\pi}{3} \)  
   c. \( \frac{\pi}{3} - \frac{\pi}{4} \)  
   d. \( \frac{\pi}{3} + \frac{\pi}{4} \)  
   e. \( \frac{\pi}{3} - \frac{\pi}{4} \)

5. Use three-digit rounding arithmetic to perform the following calculations. Compute the absolute error and relative error with the exact value determined to at least five digits.
   
   a. \( 133 + 0.921 \)  
   b. \( 133 - 0.499 \)  
   c. \( (121 - 0.327) - 119 \)
   d. \( (121 - 119) - 0.327 \)  
   e. \( \frac{15 \pi}{10} - \frac{8}{3} \)  
   f. \( \frac{15 \pi}{10} - \frac{8}{3} - 5.4 \)
   g. \( \frac{\pi}{3} (\frac{\pi}{3}) \)  
   h. \( \frac{\pi}{3} \)

6. Repeat Exercise 5 using four-digit rounding arithmetic.

7. Repeat Exercise 5 using three-digit chopping arithmetic.

8. Repeat Exercise 5 using four-digit chopping arithmetic.

9. The first three nonzero terms of the Maclaurin series for the arctangent function are \( x -(1/3)x^3 + (1/5)x^5 \). Compute the absolute error and relative error in the following approximations of \( x \) using the polynomial in place of the arctangent.
   
   a. \( 4[\arctan(\frac{1}{3}) + \arctan(\frac{1}{3})] \)  
   b. \( 16\arctan(\frac{1}{3}) - 4\arctan(\frac{1}{3}) \)

10. The number \( e \) can be defined by \( e = \sum_{n=0}^{\infty} (1/n!) \). Compute the absolute error and relative error in the following approximations of \( e \):
11. Let
\[ f(x) = \frac{x \cos x - \sin x}{x - \sin x} \]
a. Find \( \lim_{x \to 0} f(x) \).
b. Use four-digit rounding arithmetic to evaluate \( f(0.1) \).
c. Replace each trigonometric function with its third Maclaurin polynomial, and repeat part (b).
d. The actual value is \( f(0.1) = -1.99899998 \). Find the relative error for the values obtained in parts (b) and (c).

12. Let
\[ f(x) = \frac{e^x - e^{-x}}{x} \]
a. Find \( \lim_{x \to 0} \left( e^x - e^{-x} \right)/x \).
b. Use three-digit rounding arithmetic to evaluate \( f(0.1) \).
c. Replace each exponential function with its third Maclaurin polynomial, and repeat part (b).
d. The actual value is \( f(0.1) = 2.003335000 \). Find the relative error for the values obtained in parts (b) and (c).

13. Use four-digit rounding arithmetic and the formulas of Example 5 to find the most accurate approximations to the roots of the following quadratic equations. Compute the absolute errors and relative errors.
   a. \( \frac{1}{3} x^2 - \frac{123}{4} x + \frac{1}{6} = 0 \)
   b. \( \frac{1}{3} x^2 + \frac{123}{4} x - \frac{1}{6} = 0 \)
   c. \( 1.002x^2 - 11.01x + 0.01265 = 0 \)
   d. \( 1.002x^2 + 11.01x + 0.01265 = 0 \)

14. Repeat Exercise 13 using four-digit chopping arithmetic.

15. Use the 64-bit long real format to find the decimal equivalent of the following floating-point machine numbers.
   a. \( 0 \ 10000001010 \ 100100110000000000000000000000000000000000 \)
   b. \( 1 \ 10000001010 \ 100100110000000000000000000000000000000000 \)
   c. \( 0 \ 01111111111 \ 010001100000000000000000000000000000000000 \)
   d. \( 0 \ 01111111111 \ 010001100000000000000000000000000000000000 \)

16. Find the next largest and smallest machine numbers in decimal form for the numbers given in Exercise 15.

17. Suppose two points \((x_0, y_0)\) and \((x_1, y_1)\) are on a straight line with \( y_1 \neq y_0 \). Two formulas are available to find the \( x \)-intercept of the line:
\[ x = \frac{x_0y_1 - x_1y_0}{y_1 - y_0} \quad \text{and} \quad x = x_0 - \frac{(x_1 - x_0)y_0}{y_1 - y_0} \]
a. Show that both formulas are algebraically correct.
b. Use the data \((x_0, y_0) = (1.31, 3.24)\) and \((x_1, y_1) = (1.93, 4.76)\) and three-digit rounding arithmetic to compute the \( x \)-intercept both ways. Which method is better and why?