Chapter 2

Numerical Computing

We introduce the computer numbers typically available on modern computers and discuss the quality of the approximations produced by numeric computing. The chapter ends with a number of simple examples that illustrate some of the pitfalls of numerical computing.

2.1 Numbers

Most scientific programming languages provide users with two types of numbers: integers and floating-point numbers. (The floating-point numbers are a subset of the real numbers).

Computer numbers are stored as a sequence of bits. Recall that a bit assumes one of two values, 0 or 1. So, if the number $v$ is stored using 8 bits, then we may write it as

$$\text{stored}(v) = b_7b_6b_5b_4b_3b_2b_1b_0$$

Here, $b_i$ represents the $i^{th}$ bit of $v$; general practice dictates that the indices assigned to these bits start at 0 and increase as we proceed to the left. Because each bit can assume only one of the values 0 and 1, there are $2^8 = 256$ distinct patterns of bits associated with an 8-bit number. However, the value that is associated with each of these patterns depends on what kind of number $v$ represents.

2.1.1 Integers

The most common type of integers are the **signed integers**. Most computers store signed integers using *two’s complement notation*. Using this notation, the 8-bit signed integer $i$, stored as

$$\text{stored}(i) = b_7b_6b_5b_4b_3b_2b_1b_0$$

is assigned the value

$$\text{value}(i) = b_7(-2^7) + b_6(2^6) + \cdots + b_0(2^0)$$

Among 8-bit signed integers, the smallest value is $-128$ and the largest is $127$. This asymmetry arises because the total number of possible bit patterns, $2^8 = 256$, is even. Since one bit pattern is assigned to zero, there remain an odd number of patterns to be divided between the representations of the positive and negative integers.

The 8-bit **unsigned integer** $j$ stored as

$$\text{stored}(j) = b_7b_6b_5b_4b_3b_2b_1b_0$$

is assigned the value

$$\text{value}(j) = b_7(2^7) + b_6(2^6) + b_5(2^5) + \cdots + b_0(2^0)$$

Among 256 8-bit unsigned integers, the smallest value is 0 and the largest is 255.
Problem 2.1.1. Determine a formula for the smallest and the largest \( n \)-bit unsigned integers. What are the numerical values of the smallest and the largest \( n \)-bit unsigned integers for each of \( n = 11, n = 16, n = 32 \) and \( n = 64 \)?

Problem 2.1.2. Determine a formula for the smallest and the largest \( n \)-bit signed integers. What are numerical values of the smallest and the largest \( n \)-bit signed integers for each of \( n = 11, n = 16, n = 32 \) and \( n = 64 \)?

Problem 2.1.3. List the value of each 8-bit signed integer for which the negative of this value is not also an 8-bit signed integer.

### 2.1.2 Floating-Point Numbers

In early computers the types of floating-point numbers available depended not only on the programming language but also on the computer. The situation changed dramatically after 1985 with the widespread adoption of the ANSI/IEEE 754-1985 Standard for Binary Floating-Point Arithmetic. For brevity we’ll call this the Standard. It defines several types of floating-point numbers. The most widely implemented is its 64-bit double precision (DP) type. In most computer languages the programmer can also use the 32-bit single precision (SP) type of floating-point numbers. If a program uses just one type of floating-point number, we refer to that type as the working precision (WP) floating-point numbers. (So, WP will usually be either DP or SP.)

The important quantity known as the machine epsilon, \( \epsilon \), or more precisely \( \epsilon_{wp} \), is the distance from 1 to the next larger number in the working precision.

### Double Precision Numbers

An important property of double precision (DP) numbers is that the computed result of every sum, difference, product, or quotient of a pair of DP numbers is another valid DP number. For this property to be possible, the DP numbers in the Standard include representations of finite real numbers as well as the special numbers \( \pm \infty \) and NaN (Not-a-Number). For example, the value of the DP quotient \( 1/0 \) is the special number \( \infty \) and the value of the DP product \( \infty \times 0 \), which is mathematically essentially undefined, has the value NaN. Now, we focus on the normalized DP numbers and defer further description of the special numbers to a more advanced course.

In the Standard, a DP number \( y \) is stored as

\[
\text{stored}(y) = \begin{array}{c}
\text{s}(y) & \text{e}(y) & \text{f}(y) \\
\hline
b_{63} & b_{62}b_{61} \cdots b_{52} & b_{51}b_{50} \cdots b_{0}
\end{array}
\]

The boxes illustrate how the 64 bits are partitioned into 3 fields: a 1-bit unsigned integer \( s(y) \), the sign bit of \( y \), an 11-bit unsigned integer \( e(y) \), the biased exponent of \( y \), and a 52-bit unsigned integer \( f(y) \), the fraction of \( y \). Because the sign of \( y \) is stored explicitly, with \( s(y) = 0 \) for positive and \( s(y) = 1 \) for negative values of \( y \), each DP number can be negated. The 11-bit unsigned integer \( e(y) \) represents values from 0 through 2047:

- If \( e(y) = 2047 \), then \( y \) is a special number.
- If \( 0 < e(y) < 2047 \), then \( y \) is a normalized floating-point number with value

\[
\text{value}(y) = (-1)^{s(y)} \cdot \left\{ 1 + \frac{f(y)}{2^52} \right\} \cdot 2^{E(y)}
\]

where \( E(y) = e(y) - 1023 \) is the (unbiased) exponent of \( y \).

- If \( e(y) = 0 \) and \( f(y) \neq 0 \), then \( y \) is a denormalized floating-point number with value

\[
\text{value}(y) = (-1)^{s(y)} \cdot \left\{ 0 + \frac{f(y)}{2^52} \right\} \cdot 2^{-1022}
\]

Here the exponent of \( y \) is \( E(y) = -1022 \).
2.1. NUMBERS

- If \( e(y) = 0 \) and \( f(y) = 0 \), then \( y \) has value zero.

The **significant** of \( y \) is \( 1 + \frac{f(y)}{2^{52}} \) for normalized numbers, \( 0 + \frac{f(y)}{2^{52}} \) for denormalized numbers, and 0 for zero. The normalized DP numbers \( y \) with exponent \( E(y) = k \) belong to **binade** \( k \). For example, binade \(-1\) consists of the numbers \( y \) whose magnitudes are in the half open interval \( \left[ \frac{1}{2}, 1 \right) \), binade 0 consists of the numbers \( y \) for which \( |y| \in [1, 2) \), and binade 1 consists of the numbers \( y \) for which \( |y| \in [2, 4) \). In general, binade \( k \) consists of the numbers \( y \) for which \( 2^k \leq |y| < 2^{k+1} \). Each DP binade contains the same number of positive DP numbers. We define the **DP machine epsilon** \( \varepsilon_{dp} \) as the distance from 1 to the next larger DP number. For a finite nonzero DP number \( y \), we define \( \text{ulp}_{dp}(y) \equiv \varepsilon_{dp}2^{E(y)} \) as the **DP unit-in-the-last-place** of \( y \).

**Problem 2.1.4.** What are the smallest and largest possible values of the significant of a normalized DP number?

**Problem 2.1.5.** What are the smallest and largest positive normalized DP numbers?

**Problem 2.1.6.** How many DP numbers are located in each binade? Sketch enough of the positive real number line so that you can place marks at the endpoints of each of the DP binades \(-3, -2, -1, 0, 1, 2 \) and 3. What does this sketch suggest about the spacing between successive positive DP numbers?

**Problem 2.1.7.** Show that \( \varepsilon_{dp} = 2^{-52} \).

**Problem 2.1.8.** Show that \( y = q \cdot \text{ulp}_{dp}(y) \) where \( q \) is an integer with \( 1 \leq |q| \leq 2^{53} - 1 \).

**Problem 2.1.9.** For a normalized DP number \( y \), show that

\[
1 \leq \frac{|y| \cdot \varepsilon_{dp}}{\text{ulp}_{dp}(y)} < 2
\]

Remark: This result suggests that \( \text{ulp}_{dp}(y) \) can be approximated by \( |y| \cdot \varepsilon_{dp} \).

**Problem 2.1.10.** If \( y \) is a normalized DP number in binade \( k \), then what is the distance to the next larger DP number? Express your answer in terms of \( \varepsilon_{dp} \).

**Problem 2.1.11.** Let \( y \) be a DP number. What is the fractional part of the significant of \( y \)? Express your answer in terms of the quantities \( s(y) \), \( f(y) \), and \( E(y) \). Remark: As an example, the real number 34.156 has whole part 34 and fractional part 0.156.

**Problem 2.1.12.** Show that neither \( y = \frac{1}{3} \) nor \( y = \frac{1}{10} \) is a DP number. Hint: If \( y \) is a DP number, then \( 2^m y \) is a 53-bit unsigned integer for some (positive or negative) integer \( m \).

**Problem 2.1.13.** What is the largest positive integer \( n \) such that \( 2^n - 1 \) is a DP number?

**Single Precision Numbers**

In the Standard, a single precision (SP) number \( x \) is stored as

\[
\text{stored}(x) = \boxed{s(x)} \quad \boxed{e(x)} \quad \boxed{f(x)}
\]

The boxes illustrate how the 32 bits are partitioned into 3 fields: a 1-bit unsigned integer \( s(x) \), the **sign bit** of \( x \), an 8-bit unsigned integer \( e(x) \), the **biased exponent** of \( x \), and a 23-bit unsigned integer \( f(x) \), the **fraction** of \( x \). The sign bit \( s(x) = 0 \) for positive and \( s(x) = 1 \) for negative SP numbers. The 8-bit unsigned integer \( e(x) \) represents values in the range from 0 through 255:

- If \( e(x) = 255 \), then \( x \) is a special number.
CHAPTER 2. NUMERICAL COMPUTING

- If $0 < e(x) < 255$, then $x$ is a normalized floating-point number with value
  \[ \text{value}(x) = (-1)^{s(x)} \cdot \left\{ 1 + \frac{f(x)}{2^{23}} \right\} \cdot 2^{E(x)} \]
  where $E(x) \equiv e(x) - 127$ is the (unbiased) exponent of $x$.

- If $e(x) = 0$ and $f(x) \neq 0$, then $x$ is a denormalized floating-point number with value
  \[ \text{value}(x) = (-1)^{s(x)} \cdot \left\{ 0 + \frac{f(x)}{2^{23}} \right\} \cdot 2^{-126} \]
  Here the exponent of $x$ is $E(x) = -126$.

- If both $e(x) = 0$ and $f(x) = 0$, then $x$ is zero.

The significand of $x$ is $1 + \frac{f(x)}{2^{23}}$ for normalized numbers, $0 + \frac{f(x)}{2^{23}}$ for denormalized numbers, and 0 for zero. The normalized SP numbers $y$ with exponent $E(y) = k$ belong to binade $k$. For example, binade $-1$ consists of the numbers $y$ whose magnitudes are in the half open interval $[\frac{1}{2}, 1)$, binade 0 consists of the numbers $y$ for which $|y| \in [1, 2)$, and binade 1 consists of the numbers $y$ for which $|y| \in [2, 4)$. In general, binade $k$ consists of the numbers $y$ for which $2^k \leq |y| < 2^{k+1}$. We define the SP machine epsilon $\epsilon_{sp}$ as the distance from 1 to the next larger SP number. For a finite nonzero (normalized or denormalized) SP number $x$, we define $\text{ulp}_{\text{sp}}(x) \equiv \epsilon_{sp} 2^{E(x)}$ as the SP unit-in-the-last-place of $x$.

**Problem 2.1.14.** What are the smallest and largest possible values of the significand of a normalized SP number?

**Problem 2.1.15.** What are the smallest and largest positive normalized SP numbers?

**Problem 2.1.16.** How many SP numbers are located in each binade? Sketch enough of the positive real number line so that you can place marks at the endpoints of each of the SP binades $-3$, $-2$, $-1$, 0, 1, 2 and 3. Each SP binade contains the same number of positive SP numbers. What does this sketch suggest about the spacing between consecutive positive SP numbers?

**Problem 2.1.17.** Let $x$ be a normalized positive SP number. Write $x$ as a normalized DP number.

**Problem 2.1.18.** Compare the sketches produced in Problems 2.1.6 and 2.1.16. How many DP numbers lie between consecutive positive SP numbers? We say that distinct real numbers $x$ and $y$ are resolved by SP numbers if the SP number nearest $x$ is not the same as the SP number nearest $y$. Write down the analogous statement for DP numbers. Given two distinct real numbers $x$ and $y$, are $x$ and $y$ more likely to be resolved by SP numbers or by DP numbers? Justify your answer.

**Problem 2.1.19.** Show that $\epsilon_{sp} = 2^{-23}$.

**Problem 2.1.20.** Show that $x = p \cdot \text{ulp}_{\text{sp}}(x)$ where $p$ is an integer with $1 \leq |p| \leq 2^{24} - 1$.

**Problem 2.1.21.** For a normalized SP number $x$, show that
  \[ 1 \leq \frac{|x| \cdot \epsilon_{sp}}{\text{ulp}_{\text{sp}}(x)} < 2 \]
  Remark: This result suggests that $\text{ulp}_{\text{sp}}(x)$ can be approximated by $|x| \cdot \epsilon_{sp}$.

**Problem 2.1.22.** If $x$ is a normalized SP number in binade $k$, what is the distance to the next larger SP number? Express your answer in terms of $\epsilon_{sp}$.

**Problem 2.1.23.** Let $x$ be a SP number. What is the fractional part of the significand of $x$? Express your answer in terms of the quantities $s(x)$, $f(x)$ and $E(x)$. Remark: As an example, the real number 34.156 has whole part 34 and fractional part 0.156.

**Problem 2.1.24.** Show that neither $x = \frac{1}{3}$ nor $x = \frac{1}{10}$ is a SP number. Hint: If $x$ is a SP number, then $2^m x$ is an integer for some (positive or negative) integer $m$.

**Problem 2.1.25.** What is the largest positive integer $n$ such that $2^n - 1$ is a SP number?
2.2 Computer Arithmetic

What kind of numbers should a program use? Signed integers may be appropriate if the input data are integers and the only operations used are addition, subtraction, and multiplication. On the other hand, floating-point numbers are appropriate if the data involves fractions, has large or small magnitudes, or if the operations include division, square root, or computing transcendental functions like the sine or the logarithm.

2.2.1 Integer Arithmetic

With one exception, in integer computer arithmetic the computed sum, difference, or product of integers is the exact result. For example, the computed sum of the integers 2 and 3 is 5. The exception is when integer overflow occurs, that is when the magnitude of the exact result is too large. This happens when to represent the result of an integer arithmetic operation more bits are needed than are available in the storage area allocated for the result. Thus, for the 8-bit signed integers $i = 126$ and $j = 124$ the computed value $i + j = 250$ is a number larger than the value of any 8-bit signed integer, so it overflows. So, in the absence of integer overflow, integer arithmetic performed by the computer is exact. That you have encountered integer overflow in your program may not always be apparent. The computer may just return an incorrect integer result!

2.2.2 Floating-Point Arithmetic

The set of all integers is closed under the arithmetic operations of addition, subtraction, and multiplication. That is the sum, difference, or product of two integers is another integer. Similarly, the set of all real numbers is closed under the arithmetic operations of addition, subtraction, and multiplication, and division (except by zero). Unfortunately, the set of all DP numbers is not closed under any of the operations of add, subtract, multiply, or divide. For example, both $x = 2^{52} + 1$ and $y = 2^{52} - 1$ are DP numbers and yet their product $xy = 2^{104} - 1$ is not an DP number.

To make the DP numbers closed under the arithmetic operations of addition, subtraction, multiplication, and division, the Standard modifies slightly the result produced by each of these operations. Specifically, the Standard defines the computer arithmetic operations

$$x \oplus y = f_{\text{DP}}(x + y)$$
$$x \ominus y = f_{\text{DP}}(x - y)$$
$$x \otimes y = f_{\text{DP}}(x \times y)$$
$$x \oslash y = f_{\text{DP}}(x/y)$$

Here, $f_{\text{DP}}(z)$ is defined to be the DP number closest to the real number $z$; that is, $f_{\text{DP}}()$ is the DP rounding function\footnote{In the special case when the value $z$ is midway between two adjacent DP numbers, then $f_{\text{DP}}(z)$ is the one of these two DP numbers whose fraction is even.}. So, for example, the first equality states that $x \oplus y$, the value assigned to the sum of the DP numbers $x$ and $y$, is the DP number $f_{\text{DP}}(x + y)$. In a general sense, no approximate arithmetic on DP numbers can be more accurate than that specified by the Standard.

In summary, floating-point arithmetic is inherently approximate; the computed value of any sum, difference, product, or quotient of DP numbers is equal to the exact value rounded to the nearest floating-point DP number. In the next section we’ll discuss how to measure the quality of this approximate arithmetic.

**Problem 2.2.1.** Show that $2^{104} - 1$ is not a DP number. Hint: Recall Problem 2.1.13.

**Problem 2.2.2.** Show that each of $2^{53} - 1$ and 2 is a DP number, but that their sum is not a DP number. So, the set of all DP numbers is not closed under addition. Hint: Recall Problem 2.1.13.

**Problem 2.2.3.** Show that the set of DP numbers is not closed under subtraction; that is, find two DP numbers whose difference is not a DP number.
Problem 2.2.4. Show that the set of DP numbers is not closed under division, that is find two nonzero DP numbers whose quotient is not a DP number. Hint: Consider Problem 2.1.12.

Problem 2.2.5. The definition of the DP rounding function $f_{\text{DP}}()$ above is incomplete. Consider the set of all DP numbers, including the special numbers $\pm\infty$ and NaN. If $x$ and $y$ are the smallest positive DP numbers, then why is it reasonable that $f_{\text{DP}}(x \times y) \equiv 0$? This computation causes floating-point underflow to occur. If $x$ and $y$ are the largest positive finite DP numbers, then why is it reasonable to define $f_{\text{DP}}(x \times y) \equiv \infty$? This computation causes floating-point overflow to occur. Describe a pair DP numbers $x$ and $y$ for which it is reasonable that $f_{\text{DP}}(x \times y) \equiv \text{NaN}$.

Problem 2.2.6. Assuming floating-point underflow does not occur, why can any DP number $x$ be divided by 2 exactly? Hint: Consider the representation of $x$ as a DP number.

Problem 2.2.7. Let $x$ be a DP number. Show that $f_{\text{DP}}(x) = x$.

Problem 2.2.8. Let each of the values $x$, $y$ and $x + y$ be a DP number. What is the value of the DP number $x \oplus y$? State the extension of your result to the difference, product and quotient of DP numbers.

Problem 2.2.9. The real numbers $x$, $y$ and $z$ satisfy the associative law of addition:

$$(x + y) + z = x + (y + z)$$

Consider the DP numbers $a = -2^{60}$, $b = 2^{60}$ and $c = 2^{-60}$. Show that

$$(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$$

So, in general DP addition is not associative. Hint: Show that $b \oplus c = b$.

Problem 2.2.10. The real numbers $x$, $y$ and $z$ satisfy the associative law of multiplication:

$$(x \times y) \times z = x \times (y \times z),$$

Consider the DP numbers $a = 1 + 2^{-52}$, $b = 1 - 2^{-52}$ and $c = 1.5 + 2^{-52}$. Show that

$$(a \otimes b) \otimes c \neq a \otimes (b \otimes c)$$

So, in general DP multiplication is not associative. Hint: Show that $a \otimes b = 1$ and $b \otimes c = 1.5 - 2^{-52}$.

Problem 2.2.11. The real numbers $x$, $y$ and $z$ satisfy the distributive law:

$$x \times (y + z) = (x \times y) + (x \times z)$$

Choose values of the DP numbers $a$, $b$ and $c$ such that

$$a \otimes (b \oplus c) \neq (a \otimes b) \oplus (a \otimes c)$$

So, in general DP arithmetic is not distributive.

Problem 2.2.12. Define the SP rounding function $f_{\text{SP}}()$ that maps real numbers into SP numbers. Define the values of $x \oplus y$, $x \otimes y$, $x \otimes y$ and $x \otimes y$ for SP arithmetic.

Problem 2.2.13. Show that $2^{24} - 1$ and 2 are SP numbers, but their sum is not a SP number. So, the set of all SP numbers is not closed under addition. Hint: Recall Problem 2.1.25.
2.2. COMPUTER ARITHMETIC

2.2.3 Quality of Approximations

To illustrate the use of approximate arithmetic we employ normalized decimal scientific notation. A nonzero real number \( T \) is represented
\[
T = \pm m(T) \cdot 10^{d(T)}
\]
where \( 1 \leq m(T) < 10 \) is a real number and \( d(T) \) is a positive, negative or zero integer\(^2\). Here, \( m(T) \) is defined as the decimal significand and \( d(T) \) is defined as the decimal exponent of \( T \). For example,
\[
120 = (1.20) \cdot 10^2 \\
\pi = (3.14159\ldots) \cdot 10^0 \\
-0.01026 = (-1.026) \cdot 10^{-2}
\]
For the real number \( T = 0 \), we define \( m(T) = 1 \) and \( d(T) = -\infty \).

For any integer \( k \), decade \( k \) is the set of real numbers the value of whose exponents \( d(T) = k \). So, the decade \( k \) is the set of real numbers with values whose magnitudes are in the half open interval \([10^k, 10^{k+1})\).

For a nonzero number \( T \), its \( i^{\text{th}} \) significant digit is the \( i^{\text{th}} \) digit of \( m(T) \), counting to the right starting with the units digit. So, the units digit is the \( 1^{\text{st}} \) significant digit, the tenths digit is the \( 2^{\text{nd}} \) significant digit, the hundredths digit is the \( 3^{\text{rd}} \) significant digit, etc. For the value \( \pi \) listed above, the \( 1^{\text{st}} \) significant digit is 3, the \( 2^{\text{nd}} \) significant digit is 1, the \( 3^{\text{rd}} \) significant digit is 4, etc.

Let \( A \) approximate the true value \( T \). Frequently used measures of the error in \( A \) as an approximation to \( T \) are
\[
\text{error} = A - T \\
\text{absolute error} = |A - T| \\
\text{relative error} = \frac{A - T}{T} \\
\text{absolute relative error} = \frac{|A - T|}{T}
\]
with the relative errors being defined only when \( T \neq 0 \). The approximation \( A \) to \( T \) is \( q \)-digits accurate if the absolute error is less than \( \frac{1}{2} \) of one unit in the \( q^{\text{th}} \) significant digit of \( T \). Since \( 1 \leq m(T) < 10 \), \( A \) is a \( q \)-digits approximation to \( T \) if
\[
|A - T| \leq \frac{1}{2}m(T)|10^{d(T)}10^{-q} \leq \frac{10^{d(T)-q+1}}{2}
\]
If the absolute relative error in \( A \) is less than \( r \), then \( A \) is a \( q \)-digits accurate approximation to \( T \) provided that \( q \leq -\log_{10}(2r) \).

We can eliminate the rounding function by noting that whenever \( \text{fl}_{\text{DP}}(z) \) is a normalized DP number,
\[
\text{fl}_{\text{DP}}(z) = z(1 + \mu) \quad \text{where} \quad |\mu| \leq \frac{\epsilon_{\text{DP}}}{2}
\]
The value \( \mu \), the relative error in \( \text{fl}_{\text{DP}}(z) \), depends on the value of \( z \). Then
\[
x \oplus y = (x + y)(1 + \mu_a) \\
x \ominus y = (x - y)(1 + \mu_a) \\
x \odot y = (x \times y)(1 + \mu_m) \\
x \oslash y = (x/y)(1 + \mu_d)
\]
where the magnitude of each of the relative errors \( \mu_a, \mu_s, \mu_m \) and \( \mu_d \) is no larger than \( \frac{\epsilon_{\text{DP}}}{2} \).

\(^2\)The Standard represents numbers in binary notation. We use decimal representation to simplify the development.
Problem 2.2.14. Show that if $A$ is an approximation to $T$ with an absolute relative error less than $0.5 \cdot 10^{-16}$, then $A$ is a 16-digit accurate approximation to $T$.

Problem 2.2.15. Let $r$ be an upper bound on the absolute relative error in the approximation $A$ to $T$, and let $q$ be an integer that satisfies $q \leq -\log_{10}(2r)$. Show that $A$ is a $q$-digit approximation to $T$. Hint: Show that $|T| \leq 10^{q(T)+1}$, $r \leq \frac{10^{-q}}{2}$, and $|A - T| \leq \frac{10^{-q}T}{2} \leq \frac{10^{q(T)-q+1}}{2}$. This last inequality demonstrates that $A$ is $q$-digits accurate.

Problem 2.2.16. Let $z$ be a real number for which $f_{\text{DP}}(z)$ is a normalized DP number. Show that $|f_{\text{DP}}(z) - z|$ is at most half a DP ulp times the value of $f_{\text{DP}}(z)$. Then show that $\mu \equiv \frac{f_{\text{DP}}(z) - z}{z}$ satisfies the bound $|\mu| \leq \epsilon_{\text{DP}}$.

Problem 2.2.17. Let $x$ and $y$ be DP numbers. The relative error in $x \oplus y$ as an approximation to $x + y$ is no larger than $\frac{\epsilon_{\text{DP}}}{2}$. Show that $x \oplus y$ is about 15 digits accurate as an approximation to $x + y$. How does the accuracy change if you replace the addition operation by any one of subtraction, multiplication, or division (assuming $y \neq 0$)?

2.2.4 Propagation of Errors

There are two types of errors in any computed sum, difference, product, or quotient of numbers. The first is the error that is inherent in the numbers, and the second is the error introduced by the arithmetic.

Let $x'$ and $y'$ be DP numbers and consider computing $x' \times y'$. Commonly, $x'$ and $y'$ are the result of a previous computation. Let $x$ and $y$ be the values that $x'$ and $y'$ would have been if they had been computed exactly; that is,

$$x' = x(1 + \mu_x), \quad y' = y(1 + \mu_y)$$

where $\mu_x$ and $\mu_y$ are the relative errors in the approximations of $x'$ to $x$ and of $y'$ to $y$, respectively. Now, $x' \times y'$ is computed as $x' \otimes y'$, so

$$x' \otimes y' = f_{\text{DP}}(x' \times y') = (x' \times y')(1 + \mu_m)$$

where $\mu_m$ is the relative error in the approximation $x' \otimes y'$ of $x' \times y'$. How well does $x' \otimes y'$ approximate the exact result $x \times y$? We find that

$$x' \otimes y' = (x \times y)(1 + \mu)$$

where

$$\mu = (1 + \mu_x)(1 + \mu_y)(1 + \mu_m) - 1$$

Expanding the products

$$\mu = \mu_x + \mu_y + \mu_m + \mu_x\mu_y + \mu_x\mu_m + \mu_y\mu_m + \mu_x\mu_y\mu_m$$

Generally, we can drop the terms involving products of the $\mu$’s because the magnitude of each value $\mu$ is small relative to 1, so the magnitudes of products of the $\mu$’s are smaller still. Consequently

$$\mu \approx \mu_x + \mu_y + \mu_m$$

So the relative error in $x' \otimes y'$ is (approximately) equal to the sum of

a. the relative errors inherent in each of the values $x'$ and $y'$

b. the relative error introduced when the values $x'$ and $y'$ are multiplied
In an extended computation, we expect the error in the final result to come from the (hopefully slow) accumulation of the errors in the initial data and of the errors from the arithmetic operations on that data. This is a major reason why DP arithmetic is mainly used in general scientific computation. While the final result may be represented adequately by a SP number, we use DP arithmetic in an attempt to reduce the effect of the accumulation of errors introduced by the computer arithmetic because the relative errors \( \mu \) in DP arithmetic are so much smaller than in SP arithmetic.

Relatively few DP computations produce exactly a DP number. However, subtraction of nearly equal DP numbers of the same sign is always exact and so is always a DP number. This exact cancellation result is stated mathematically as follows:

\[
x' \oplus y' = x' - y'
\]

whenever \( \frac{1}{2} \leq \frac{x'}{y'} \leq 2 \). So, in this case \( \mu_x = 0 \) and we expect that \( \mu \approx \mu_x + \mu_y \). However, it is easy to see that

\[
x' \oplus y' = x' - y' = (x - y)(1 + \mu)
\]

where \( \mu = \frac{x \mu_x - y \mu_y}{x - y} \). We obtain an upper bound on this relative error \( \mu \) by applying the triangle inequality:

\[
\frac{|\mu|}{|\mu_x| + |\mu_y|} \leq g = \frac{|x| + |y|}{|x - y|}
\]

The left hand side measures how the relative errors \( \mu_x \) and \( \mu_y \) in the values \( x' \) and \( y' \), respectively, are magnified to produce the relative error \( \mu \) in the value \( x' - y' \). The right hand side, \( g \), is an upper bound on this magnification. Observe that \( g \geq 1 \) and that \( g \) grows as \( |x - y| \) gets smaller, that is as more cancellation occurs in computing \( x - y \). When \( g \) is large, the relative error in \( x' - y' \) may be large. And, when the relative error in \( x' - y' \) is large, catastrophic cancellation has occurred. In the next section we present several examples where computed results suffer from the effect of catastrophic cancellation.

**Problem 2.2.18.** Derive the expression

\[
x' \oplus y' = x' - y' = (x - y)(1 + \mu)
\]

where \( \mu = \frac{x \mu_x - y \mu_y}{x - y} \). Show that

\[
|x \mu_x - y \mu_y| \leq |x| |\mu_x| + |y| |\mu_y| \leq (|x| + |y|)(|\mu_x| + |\mu_y|)
\]

and that

\[
\frac{|x \mu_x - y \mu_y|}{x - y} \leq (|\mu_x| + |\mu_y|) \frac{|x| + |y|}{|x - y|}
\]

### 2.3 Examples

The following examples are intended to illustrate some of the less obvious pitfalls in simple scientific computations.

#### 2.3.1 Plotting a Polynomial

Consider the polynomial \( p(x) = (1 - x)^{10} \), which can be written in power series form as

\[
p(x) = x^{10} - 10x^9 + 45x^8 - 120x^7 + 210x^6 - 252x^5 + 210x^4 - 120x^3 + 45x^2 - 10x + 1
\]

Suppose we use this power series form to evaluate and plot \( p(x) \) on the interval \([0.99, 1.01]\) using 103 equally spaced points, that is at a spacing of 0.01. If DP arithmetic is used to evaluate the polynomial at these points, and the resulting values are plotted, we obtain a graph such as the one shown in
Fig. 2.1. This plot suggests that \( p(x) \) has many zeros in the interval \([0.99, 1.01]\). (Remember, if \( p(x) \) changes sign at two points then, by continuity, \( p(x) \) has a zero somewhere between those points.) However, the factored form \( p(x) = (1 - x)^{10} \) implies there is only a single zero, of multiplicity 10, at the point \( x = 1 \). Roundoff error incurred while evaluating the power series form of \( p(x) \) produces this inconsistency.

![Plot of the power form of \( p(x) \equiv (1 - x)^{10} \) evaluated in DP arithmetic](image)

Figure 2.1: Plot of the power form of \( p(x) \equiv (1 - x)^{10} \) evaluated in DP arithmetic

Note that in Fig. 2.1, the maximum amplitudes of the oscillations are larger to the right of \( x = 1 \) than to the left. Recall that \( x = 1 \) is the boundary between binade \(-1\) and binade \( 0 \). As a result, the magnitude of the maximum error incurred by rounding can be a factor of 2 larger to the right of \( x = 1 \) than to the left, which is essentially what we observe.

### 2.3.2 Repeated Square Roots

Let \( n \) be a positive integer and let \( x \) be a DP number such that \( 1 \leq x < 4 \). Consider the following procedure. First, initialize the DP variable \( t \) to the value of \( x \). Next, take the square root of \( t \) a total of \( n \) times via the assignment \( t := \sqrt{t} \). Then, square the resulting value \( t \) a total of \( n \) times via the assignment \( t := t \ast t \). Finally, print the value of error, \( t - x \). The computed results of such an experiment for various values of \( x \), using \( n = 100 \), are shown in Table 2.1.

Note that, but for rounding, the value of the error in the third column would be zero, but the results show errors much larger than zero! To understand what is happening here, observe that taking the square root of a number discards information. The square root function maps the interval \([1, 4]\) onto the binade \([1, 2]\). Now \([1, 4]\) is the union of the two binades \([1, 2]\) and \([2, 4]\). Furthermore, each binade contains \( 2^{53} \) DP numbers. So, the square root function maps each of \( 2 \cdot 2^{53} = 2^{54} \) arguments into one of \( 2^{53} \) possible square roots. On average, then, the square root function maps two DP numbers in \([1, 4]\) into one DP number in \([1, 2]\). So, generally, the DP square root of a DP argument does not contain sufficient information to recover that DP argument, that is taking the DP square root of a DP number usually loses information.
### 2.3. EXAMPLES

<table>
<thead>
<tr>
<th>$x$</th>
<th>$t$</th>
<th>$x-t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>1.2500</td>
<td>1.0000</td>
<td>0.2500</td>
</tr>
<tr>
<td>1.5000</td>
<td>1.0000</td>
<td>0.5000</td>
</tr>
<tr>
<td>1.7500</td>
<td>1.0000</td>
<td>0.7500</td>
</tr>
<tr>
<td>2.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>2.2500</td>
<td>1.0000</td>
<td>1.2500</td>
</tr>
<tr>
<td>2.5000</td>
<td>1.0000</td>
<td>1.5000</td>
</tr>
<tr>
<td>2.7500</td>
<td>1.0000</td>
<td>1.7500</td>
</tr>
<tr>
<td>3.0000</td>
<td>1.0000</td>
<td>2.0000</td>
</tr>
<tr>
<td>3.2500</td>
<td>1.0000</td>
<td>2.2500</td>
</tr>
</tbody>
</table>

Table 2.1: Results of the repeated square root experiment. Here $x$ is the exact value, $t$ is the computed result (which in exact arithmetic should be the same as $x$), and $x-t$ is the absolute error.

#### 2.3.3 Estimating the Derivative

Consider the forward difference estimate

$$
\Delta_h f(x) \equiv \frac{f(x + h) - f(x)}{h}
$$

of the derivative $f'(x)$ of a continuously differentiable function $f(x)$. For small values of the increment $h$, Taylor series gives

$$
\Delta_h f(x) \approx f'(x) + \frac{hf''(x)}{2}
$$

assuming the second derivative $f''(x)$ also exists and is continuous in an interval containing both $x$ and $x+h$. As the increment $h \to 0$ the value of $\Delta_h f(x) \to f'(x)$, a fact familiar from calculus.

Suppose, when computing $\Delta_h f(x)$, that the only errors that occur involve when rounding the exact values of $f(x)$ and $f(x+h)$ to working precision (WP) numbers, that is

$$
\bar{f}_{WP}(f(x)) = f(x)(1 + \mu_1), \quad \bar{f}_{WP}(f(x+h)) = f(x+h)(1 + \mu_2)
$$

where $\mu_1$ and $\mu_2$ account for the relative errors in the rounding. If $\Delta_{WP} f(x)$ denotes the computed value of $\Delta_h f(x)$, then

$$
\Delta_{WP} f(x) = \frac{\bar{f}_{WP}(f(x+h)) - \bar{f}_{WP}(f(x))}{h} = \frac{f(x+h)(1 + \mu_2) - f(x)(1 + \mu_1)}{h} \\
= \Delta_h f(x) + \frac{f(x+h)\mu_2 - f(x)\mu_1}{h} \\
\approx f'(x) + \frac{1}{2} hf''(x) + \frac{f(x+h)\mu_2 - f(x)\mu_1}{h}
$$

where each relative error $|\mu_i| \leq \frac{\epsilon_{WP}}{2}$ and $\epsilon_{WP}$ is the WP machine epsilon. (So, for SP arithmetic $\epsilon_{WP} = 2^{-23}$ and for DP arithmetic $\epsilon_{WP} = 2^{-52}$.) Hence we obtain the following bound on the absolute relative error

$$
r \equiv \left| \frac{\Delta_{WP} f(x) - f'(x)}{f'(x)} \right| \approx h \left| \frac{f''(x)}{2f'(x)} \right| + \frac{1}{h} \left| \frac{f(x+h)\mu_2 - f(x)\mu_1}{f'(x)} \right| = c_1 h + \frac{c_2}{h} \equiv R
$$

in accepting $\Delta_{WP} f(x)$ as an approximation of the value $f'(x)$. (The values $c_1 \equiv \left| \frac{f''(x)}{2f'(x)} \right|$ and $c_2 \equiv \left| \frac{f(x)\epsilon_{WP}}{f'(x)} \right|$ are independent of the increment $h$.) In deriving this upper bound on $r$ we have assumed that $f(x+h) \approx f(x)$ which is valid when $h$ is small and $f(x)$ is continuous.
Consider the expression for the upper bound $R$. The first term $c_1 h \to 0$ as $h \to 0$, accounting for the error in accepting $\Delta_h f(x)$ as an approximation of $f'(x)$. The second term $\frac{c_2}{h} \to \infty$ as $h \to 0$, accounting for the error arising from using the computed, rather than the exact, values of $f(x)$ and $f(x + h)$. So, as $h \to 0$, we might expect the absolute relative error in the forward difference estimate $\Delta_{wp} f(x)$ of the derivative first to decrease and then to increase.

As an illustration of the usefulness of this bound, consider using SP arithmetic, the function $f(x) = \sin(x)$ with $x = 1$ radian, and the sequence of increments $h \equiv 2^{-n}$ for $n = 1, 2, 3, \ldots, 22$. Fig. 2.2 uses dots to display $-\log_{10}(2r)$, the digits of accuracy in the computed forward difference estimate of the derivative. The figure also uses a solid curve to display the corresponding values of $-\log_{10}(2R)$, an estimate of the minimum digits of accuracy obtained from the model of rounding. Observe that the forward difference estimate becomes more accurate as the curve increases, reaching a maximum accuracy of a little better than 4 digits at $n = 12$, and then becomes less accurate as the curve decreases. The maximum accuracy (that is the minimum value of $R$) predicted by the model is proportional to the square root of the working-precision.

**Problem 2.3.1.** Let $R(h) \equiv c_1 h + \frac{c_2}{h}$ where $c_1$, $c_2$ are positive. For what value of $h$ does $R(h)$ attain its smallest value? What is this smallest value? In the expression for the smallest value of $R(h)$, substitute $c_1 \equiv \frac{f''(x)}{2f'(x)}$ and $c_2 \equiv \frac{f(x)e_{wp}}{f'(x)}$. Hence, show that the smallest value depends on $\sqrt{e_{wp}}$. If $h = 2^{-n}$, what value of $n$ corresponds most closely to the value of $h$ that gives the minimum value of $R(h)$?

### 2.3.4 A Recurrence Relation

Consider the sequence of values $\{V_j\}_{j=0}^{\infty}$ defined by the definite integrals

$$V_j = \int_0^1 e^{j-1}x^j \, dx, \quad j = 0, 1, 2, \ldots$$
### 2.3. EXAMPLES

<table>
<thead>
<tr>
<th>( j )</th>
<th>( \hat{V}_j )</th>
<th>( \frac{\hat{V}_j}{j!} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6.3212E-01</td>
<td>6.3212E-01</td>
</tr>
<tr>
<td>1</td>
<td>3.6788E-01</td>
<td>3.6788E-01</td>
</tr>
<tr>
<td>2</td>
<td>2.6424E-01</td>
<td>1.3212E-01</td>
</tr>
<tr>
<td>3</td>
<td>2.0728E-01</td>
<td>3.4546E-02</td>
</tr>
<tr>
<td>4</td>
<td>1.7089E-01</td>
<td>7.1205E-03</td>
</tr>
<tr>
<td>5</td>
<td>1.4553E-01</td>
<td>1.2128E-03</td>
</tr>
<tr>
<td>6</td>
<td>1.2680E-01</td>
<td>1.7611E-04</td>
</tr>
<tr>
<td>7</td>
<td>1.1243E-01</td>
<td>2.2307E-05</td>
</tr>
<tr>
<td>8</td>
<td>1.0056E-01</td>
<td>2.4941E-06</td>
</tr>
<tr>
<td>9</td>
<td>9.4933E-02</td>
<td>2.6161E-07</td>
</tr>
<tr>
<td>10</td>
<td>5.0674E-02</td>
<td>1.3965E-08</td>
</tr>
<tr>
<td>11</td>
<td>4.4258E-01</td>
<td>1.1088E-08</td>
</tr>
<tr>
<td>12</td>
<td>-4.3110E+00</td>
<td>-8.9999E-09</td>
</tr>
<tr>
<td>13</td>
<td>5.7043E+01</td>
<td>9.1605E-09</td>
</tr>
<tr>
<td>14</td>
<td>-7.9760E+02</td>
<td>-9.1490E-09</td>
</tr>
<tr>
<td>15</td>
<td>1.1965E+04</td>
<td>9.1498E-09</td>
</tr>
<tr>
<td>16</td>
<td>-1.9144E+05</td>
<td>-9.1498E-09</td>
</tr>
</tbody>
</table>

Table 2.2: Values of \( V_j \) determined using SP arithmetic.

A simple integration by parts demonstrates that the values \( \{V_j\}_{j=0}^\infty \) satisfy the *recurrence relation*

\[
V_j = 1 - jV_{j-1}, \quad j = 1, 2, \cdots
\]

Because we can calculate the value of

\[
V_0 = \int_0^1 e^{x-1} dx = 1 - \frac{1}{e}
\]

we can determine the values of \( V_1, V_2, \cdots \) from the recurrence starting from the computed estimate for \( V_0 \). Table 2.2 displays the values \( \hat{V}_j \) for \( j = 0, 1, \cdots, 16 \) computed by evaluating the recurrence in SP arithmetic. The first few values \( \hat{V}_j \) are positive and form a decreasing sequence. However, the values \( \hat{V}_j \) for \( j \geq 11 \) alternate in sign and increase in magnitude, contradicting the mathematical properties of the sequence. To understand why, suppose that the only rounding error that occurs is in computing \( V_0 \). So, instead of the correct initial value \( V_0 \) we have used instead the rounded initial value \( \hat{V}_0 = V_0 + \epsilon \). Let \( \{\hat{V}_j\}_{j=0}^\infty \) be the modified sequence determined exactly from the value \( \hat{V}_0 \). Using the recurrence, the first few terms of this sequence are

\[
\begin{align*}
\hat{V}_1 &= 1 - \hat{V}_0 = 1 - (V_0 + \epsilon) = V_1 - \epsilon \\
\hat{V}_2 &= 1 - 2\hat{V}_1 = 1 - 2(V_1 - \epsilon) = V_2 + 2\epsilon \\
\hat{V}_3 &= 1 - 3\hat{V}_2 = 1 - 3(V_3 + 2\epsilon) = V_3 - 3 \cdot 2\epsilon \\
\hat{V}_4 &= 1 - 4\hat{V}_3 = 1 - 4(V_4 - 3 \cdot 2\epsilon) = V_4 + 4 \cdot 3 \cdot 2\epsilon \\
&\vdots
\end{align*}
\]

(2.1)

In general, we see that

\[
\hat{V}_j = V_j + (-1)^j j! \epsilon
\]

It is reasonable to expect that the computed value \( \hat{V}_0 \) will have an absolute error \( \epsilon \) that is no larger than half a ulp in the SP value of \( V_0 \). Hence, because \( V_0 \approx 0.632 \) is in binade \(-1\), we have

\[
\epsilon \approx \frac{\text{ulp}_{\text{SP}}(V_0)}{2} = 2^{-25} \approx 3 \cdot 10^{-8}
\]

Substituting this value for \( \epsilon \) we expect that the first few terms of the computed sequence will be positive and decreasing as theory predicts. However, because
\[ j! \cdot (3 \cdot 10^{-8}) > 1 \] for all \( j \geq 11 \) and because \( V_j < 1 \) for all values of \( j \), the effect of the term \((-1)^j j!\epsilon\) in the formula for \( \tilde{V}_j \) leads us to expect that the values \( \tilde{V}_j \) will ultimately be dominated by the term \((-1)^j j!\epsilon\) and so will alternate in sign and increase in magnitude. That this analysis is reasonable is verified by the values in Table 2.2, where the error grows like \( j! \), and \(|\tilde{V}_j/j!|\) approaches a constant.

**Problem 2.3.2.** Use integration by parts to show that
\[ V_j = 1 - jV_{j-1}, \quad j = 1, 2, \ldots \]

**Problem 2.3.3.** Using the results that follow, show that the sequence \( \{V_j\}_{j=0}^{\infty} \) has positive terms and is strictly decreasing to 0.

a. Show that for \( 0 < x < 1 \) we have \( 0 < x^{j+1} < x^j \) for \( j \geq 0 \). Hence, show that \( 0 < V_{j+1} < V_j \).

b. Show that for \( 0 < x < 1 \) we have \( 0 < e^{x-1} \leq 1 \). Hence, show that \( 0 < V_j < 1/j \).

**Problem 2.3.4.** Sketch a graph of the areas described by the integrands in \( V_0, V_1, V_2 \). Argue why this shows that \( 0 < V_2 < V_1 < V_0 \). Generalize your argument to show graphically that \( 0 < V_{j+1} < V_j \) for \( j \geq 0 \).

**Problem 2.3.5.** The error in the terms of the sequence \( \{\tilde{V}_j\}_{j=0}^{\infty} \) grows because \( \tilde{V}_{j-1} \) is multiplied by \( j \). To obtain an accurate approximation of \( V_j \), we can instead run the recurrence backwards
\[ \tilde{V}_j = \frac{1 - \tilde{V}_{j+1}}{j+1}, \quad j = M - 1, M - 2, \ldots, 1, 0 \]

Now, the error in \( \tilde{V}_j \) is divided by \( j \). More specifically, if you want accurate SP approximations of the values \( V_j \) for \( 0 \leq j \leq N \), start with \( M = N + 12 \) and \( \tilde{V}_0 = 0 \) and compute the values of \( \tilde{V}_j \) for all \( j \) in \( j = M - 1, M - 2, \ldots, 1, 0 \). We start 12 terms beyond the first value of \( \tilde{V}_N \) so that the error associated with using \( \tilde{V}_{N+12} = 0 \) will be divided by at least \( 12! \approx 4.8 \cdot 10^8 \), a factor large enough to make the contribution of the initial error in \( V_M \) to the error in \( \tilde{V}_N \) less than a SP ulp in \( V_N \). (We know that \( V'_M \) is in error - from Problem 2.3.3 \( V_M \) is actually a small positive number, \( 0 < V_M < 1/M \), so \( |V'_M - V_M| < 1/M \).)

It is important to emphasize that we are not recommending the use of the recurrence as a method for evaluating the integrals \( V_j \). They may be almost trivially evaluated using integration software, such as that discussed in Chapter 5.

### 2.3.5 Summing the Exponential Series

The Taylor series for the exponential function
\[ e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots \]
converges for all values of \( x \) both positive and negative. Let the term
\[ T_n = \frac{x^n}{n!} \]
and define the partial sum
\[ S_n = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^{n-1}}{(n-1)!} \]
so that \( S_{n+1} = S_n + T_n \).
### 2.3. EXAMPLES

<table>
<thead>
<tr>
<th>$n$</th>
<th>$T_n$</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0000e+00</td>
<td>1.0000E+00</td>
</tr>
<tr>
<td>5</td>
<td>-3.0171e+04</td>
<td>-2.4057e+04</td>
</tr>
<tr>
<td>10</td>
<td>3.6122e+06</td>
<td>2.4009e+06</td>
</tr>
<tr>
<td>15</td>
<td>-3.6292e+07</td>
<td>-2.0703e+07</td>
</tr>
<tr>
<td>20</td>
<td>7.0624e+07</td>
<td>3.5307e+07</td>
</tr>
<tr>
<td>25</td>
<td>-4.0105e+07</td>
<td>-1.7850e+07</td>
</tr>
<tr>
<td>30</td>
<td>8.4909e+06</td>
<td>3.4061e+06</td>
</tr>
<tr>
<td>35</td>
<td>-7.8914e+05</td>
<td>-2.8817e+05</td>
</tr>
<tr>
<td>40</td>
<td>3.6183e+04</td>
<td>1.2126e+04</td>
</tr>
<tr>
<td>45</td>
<td>-8.9354e+02</td>
<td>-2.7673e+02</td>
</tr>
<tr>
<td>50</td>
<td>1.2724e+01</td>
<td>3.6627e+00</td>
</tr>
<tr>
<td>55</td>
<td>-1.1035e-01</td>
<td>-2.9675e-02</td>
</tr>
<tr>
<td>60</td>
<td>6.0962e-04</td>
<td>1.5382e-04</td>
</tr>
<tr>
<td>65</td>
<td>-2.2267e-06</td>
<td>-5.2412e-07</td>
</tr>
<tr>
<td>70</td>
<td>5.5509e-09</td>
<td>6.2893e-09</td>
</tr>
<tr>
<td>75</td>
<td>-9.7035e-12</td>
<td>5.0406e-09</td>
</tr>
<tr>
<td>80</td>
<td>1.2178e-14</td>
<td>5.0427e-09</td>
</tr>
<tr>
<td>85</td>
<td>-1.1201e-17</td>
<td>5.0427e-09</td>
</tr>
<tr>
<td>90</td>
<td>7.6897e-21</td>
<td>5.0427e-09</td>
</tr>
<tr>
<td>95</td>
<td>-4.0042e-24</td>
<td>5.0427e-09</td>
</tr>
<tr>
<td>98</td>
<td>3.7802e-26</td>
<td>5.0427e-09</td>
</tr>
</tbody>
</table>

Table 2.3: Values of $S_n$ and $T_n$ determined using DP arithmetic.

Let’s use this series to compute $e^{-20.5} \approx 1.25 \cdot 10^{-9}$. For $x = -20.5$, the terms in this series alternate in sign and their magnitudes increase until we reach the term $T_{20} \approx 7.06 \cdot 10^7$ and then the terms alternate in sign and their magnitudes steadily decrease to zero. So, for any value $n > 20$, the theory of alternating series tells us that

$$|S_n - e^{-20.5}| < |T_n|$$

that is the absolute error in the partial sum $S_n$ as an approximation to the value of $e^{-20.5}$ is less than the absolute value of the first neglected term $|T_n|$. Note that DP arithmetic is about 16-digit accurate. So, if a value $n \geq 20$ is chosen so that $\frac{|T_n|}{e^{-20.5}} < 10^{-16}$, then $S_n$ should be a 16-digit approximation to $e^{-20.5}$. The smallest value of $n$ for which this inequality is satisfied is $n = 98$, so $S_{98}$ should be a 16-digit accurate approximation to $e^{-20.5}$.

Table 2.3 displays a selection of values of the partial sums $S_n$ and the corresponding terms $T_n$ computed using DP arithmetic. From the table, it appears that the sequence of values $S_n$ is converging. However, $S_{98} \approx 5.04 \cdot 10^{-9}$, so $S_{98}$ is an approximation of $e^{-20.5}$ that is 0-digit accurate! At first glance, it appears that the computation is wrong! However, the computed value of the sum, though very inaccurate, is reasonable. Recall that DP numbers are about 16 digits accurate, so the largest term in magnitude $T_{20}$ could be in error by as much as about $|T_{20}| \cdot 10^{-16} \approx 7.06 \cdot 10^7 \cdot 10^{-16} = 7.06 \cdot 10^{-9}$. Of course, changing $T_{20}$ by this amount will change the value of the partial sum $S_{98}$ similarly. So, using DP arithmetic, it is unlikely that the computed value $S_{98}$ will provide an estimate of $e^{-20.5}$ with an absolute error much less than $7.06 \cdot 10^{-9}$.

So, how can we calculate $e^{-20.5}$ accurately using Taylor series. Clearly we must avoid the catastrophic cancellation involved in summing an alternating series with large terms when the value of the sum is a much smaller number. There follow two approaches which exploit simple mathematical properties of the exponential function.

1. Consider the relation $e^{-x} = \frac{1}{e^x}$. If we use a Taylor series for $e^x$ for $x = 20.5$ it still involves large and small terms $T_n$ but they are all positive and there is no cancellation in evaluating
the sum $S_n$. In fact the terms $T_n$ are the absolute values of those appearing in Table 2.3. In adding this sum we continue until adding further terms can have no impact on the sum. When the terms are smaller then $e^{20.5} \cdot 10^{-16} \approx 8.00 \cdot 10^8 \cdot 10^{-16} = 8.00 \cdot 10^{-8}$ they have no further impact so we stop at the first term smaller than this. Forming this sum and stopping when $|T_n| < 8.00 \cdot 10^{-8}$, it turns out that we need the first 68 terms of the series to give a DP accurate approximation to $e^{20.5}$. Since we are summing a series of positive terms there is no cancellation so we expect (and get) an approximation for $e^{20.5}$ with a small relative error. Next, we compute $e^{-20.5} = \frac{1}{e^{20.5}}$ using this approximation for $e^{20.5}$. The result has at least 15 correct digits in a DP calculation.

2. Another idea is to use a form of range reduction. We can use the alternating series as long as we avoid catastrophic cancellation which leads to a large relative error. We will certainly do that if we evaluate $e^{-x}$ only for $0 < x < 1$ for in this case the magnitudes of the terms of the series are monotonically decreasing. We observe that

$$e^{-20.5} = e^{-20} e^{-0.5} = (e^{-1})^{20} e^{-0.5}$$

So if we can calculate $e^{-1}$ accurately, then raise it to the 20th power, then evaluate $e^{-0.5}$ by Taylor series, finally we can compute $e^{-20.5}$ by multiplying the results. (We “cheat” here by using the computer’s functions to evaluate $e^{-1}$ and the raise this value to the 20th power; this way these quantities are calculated accurately. If we didn’t have these functions available we’d have to find some way to compute them accurately ourselves!) Since $e^{-0.5} \approx 0.6$, to get full accuracy we terminate the Taylor series when $|T_n| < 0.6 \cdot 10^{-16}$ that is after 15 terms. This approach also delivers at least 15 digits of accuracy in a DP calculation.

**Problem 2.3.6.** Quote and prove the theorem from calculus concerning alternating series that shows that $|S_n - e^{-20.5}| < |T_n|$ in exact arithmetic.

**Problem 2.3.7.** Suppose that you use Taylor series to compute $e^x$ for a value of $x$ such that $|x| > 1$. Which is the largest term in the Taylor series? In what circumstances are there two equally sized largest terms in the Taylor series? Hint: $\frac{T_n}{T_{n-1}} = \frac{x}{n}$.

**Problem 2.3.8.** Suppose we are using SP arithmetic with an accuracy of about $10^{-7}$ and say we attempt to compute $e^{-15}$ using the alternating Taylor series. What is the largest value $T_n$ in magnitude. Using the fact that $e^{-15} \approx 3.06 \cdot 10^{-7}$ how many terms of the alternating series will be needed to compute $e^{-15}$ to full SP accuracy? What is the approximate absolute error in the sum as an approximation to $e^{-15}$? Hint: There is no need to calculate the value of the terms $T_n$ or the partial sums $S_n$ to answer this question.

**Problem 2.3.9.** For what value of $n$ does the partial sum $S_n$ form a 16-digit accurate approximation of $e^{-21.5}$? Using DP arithmetic, compare the computed value of $S_n$ with the exact value $e^{-21.5} \approx 4.60 \cdot 10^{-10}$. Explain why the computed value of $S_n$ is reasonable. Hint: Because $\frac{T_n}{T_{n-1}} = \frac{x}{n}$, if the current value of term is $T_{n-1}$, then sequence of assignment statements

```plaintext
term := x * term
term := term / n
```

converts term into the value of $T_n$.

### 2.3.6 Euclidean Length of a Vector

A commonly required computation determines the **Euclidean length**

$$p = \sqrt{a^2 + b^2}$$
of the 2-vector \( \begin{bmatrix} a \\ b \end{bmatrix} \) (The value \( p \) is the **Pythagorean sum** of \( a \) and \( b \).) Now,

\[
\min(|a|, |b|) \leq p \leq \sqrt{2} \cdot \max(|a|, |b|)
\]

so we should be able to compute \( p \) in such a way that we never encounter floating-point underflow and we rarely encounter floating-point overflow. However, when computing \( p \) via the relation \( p = \sqrt{a^2 + b^2} \) we can encounter one or both of floating-point underflow and floating-point overflow when computing the squares of \( a \) and \( b \). To avoid floating-point overflow, we can choose a value \( c \) in such a way that we can scale \( a \) and \( b \) by this value of \( c \) and the resulting scaled quantities \( \frac{a}{c} \) and \( \frac{b}{c} \) may be safely squared. Using this scaling,

\[
p = c \cdot \sqrt{\left( \frac{a}{c} \right)^2 + \left( \frac{b}{c} \right)^2}
\]

An obvious choice of scaling factor is \( c = \max(|a|, |b|) \) so that one of \( \frac{a}{c} \) and \( \frac{b}{c} \) equals 1 and the other has magnitude less than 1. Another sensible choice of scaling factor is \( c \) a power of 2 just greater than \( \max(|a|, |b|) \); the advantage of this choice is that with **Standard** arithmetic, division by 2 is performed exactly (ignoring underflow). If we choose the scaling factor \( c \) in one of these ways it is possible that the smaller squared term in that occurs when computing \( p \) will underflow but this will be essentially harmless, that is the computed value of \( p \) will be essentially correct.

Another technique that avoids both unnecessary floating-point overflow and floating-point underflow is the Moler-Morrison Pythagorean sum algorithm displayed in the pseudocode in Fig. 2.3. In this algorithm, the value \( p \) converges to \( \sqrt{a^2 + b^2} \) from below, and the value \( q \) converges rapidly to 0 from above. So, floating-point overflow can occur only if the exact value of \( p \) overflows. Also, only harmless floating-point underflow can occur.

\[
\begin{align*}
p &= \max(|a|, |b|) \\
q &= \min(|a|, |b|) \\
&\text{for } i = 1 \text{ to } N \\
r &= (q/p)^2 \\
s &= r/(4 + r) \\
p &= p + 2sp \\
q &= sq \\
&\text{next } i
\end{align*}
\]

Figure 2.3: The Moler-Morrison algorithm for computing \( p \equiv \sqrt{a^2 + b^2} \); \( N = 3 \) suffices for both SP and DP numbers.

**Problem 2.3.10.** For any values \( a \) and \( b \) show that

\[
\min(|a|, |b|) \leq p \equiv \sqrt{a^2 + b^2} \leq \sqrt{2} \cdot \max(|a|, |b|)
\]

**Problem 2.3.11.** Use DP arithmetic to compute the lengths of the vectors \( \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 10^{60} \\ 10^{61} \end{bmatrix} \) and \( \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 10^{-60} \\ 10^{-61} \end{bmatrix} \) using both the unscaled and scaled formulas for \( p \). Use the factor \( c = \max(|a|, |b|) \) in the scaled computation.

**Problem 2.3.12.** Suppose \( a = 1 \) and \( b = 10^{-60} \). If \( c = \max(|a|, |b|) \), floating-point underflow occurs in computing \( \frac{b^2}{c} \) in DP arithmetic. In this case, the Standard returns the value 0 for \( \frac{b^2}{c} \). Why is the computed value of \( p \) still accurate?
Problem 2.3.13. In the Moler-Morrison Pythagorean sum algorithm, show that though each trip through the for-loop may change the values of $p$ and $q$, in exact arithmetic it never changes the value of $p^2 + q^2$. [Remark: The combination $p^2 + q^2$ is called a loop invariant.]

Problem 2.3.14. Design a pseudocode algorithm to compute the value $d = \sqrt{x^2 + y^2 + z^2}$. If floating-point underflow occurs, it should be harmless, and floating-point overflow should occur only when the exact value of $d$ overflows.

2.3.7 Roots of a Quadratic Equation

We aim to design a reliable algorithm to determine the roots of the quadratic equation

$$ax^2 - 2bx + c = 0$$

(Note that the factor multiplying $x$ is $-2b$ and not $b$ as in the standard notation.) We assume that the coefficients $a$, $b$ and $c$ are real numbers, and that the coefficients are chosen so that the roots are real. The familiar quadratic formula for these roots is

$$x_{\pm} = \frac{b \pm \sqrt{d}}{a}$$

where $d \equiv b^2 - ac$ is the discriminant, assumed nonnegative here. The discriminant $d$ is zero if there is a double root. Here are some problems that can arise.

First, the algorithm should check for each of the special cases $a = 0$, $b = 0$ or $c = 0$. These cases are trivial to eliminate. When $a = 0$ the quadratic equation degenerates to a linear equation whose solution $\frac{c}{b}$ requires at most a division. When $b = 0$ the roots are $\pm \sqrt{-\frac{c}{a}}$. When $c = 0$ the roots are 0 and $\pm \frac{2b}{a}$. In the latter two cases the roots will normally be calculated more accurately using these special formulas than by using the quadratic formula.

Second, computing the discriminant, $d$, can lead to either underflow or overflow, typically when $b^2$ and $ac$ are computed. This problem can be eliminated by using the technique described in the previous section; simply scale the coefficients $a$, $b$ and $c$ by a power of two chosen so that $b^2$ and $ac$ can be safely computed.

Third, the computation described by the quadratic formula can suffer from catastrophic cancellation when either

- the roots are nearly equal, i.e., $ac \approx b^2$
- the roots have significantly different magnitudes, i.e., $|ac| \ll b^2$

When $ac \approx b^2$ there is catastrophic cancellation when computing $d$. This may be eliminated by computing the discriminant $d = b^2 - ac$ using higher precision arithmetic, when possible. For example, if $a$, $b$ and $c$ are SP numbers, then we can use DP arithmetic to compute $d$. If $a$, $b$ and $c$ are DP numbers, then maybe we can use QP (quadruple (extended) precision) arithmetic to compute $d$. The idea is to use a sufficiently higher precision arithmetic so that as few as possible of the digits in $b^2$ and $ac$ are discarded before the subtraction in $b^2 - ac$ is performed. For many arithmetic processors this may be achieved without user intervention. That is, the discriminant may be calculated to higher precision automatically, by computing the value of the whole expression $b^2 - ac$ in higher precision before rounding. When cancellation is inevitable, we can overcome much of the difficulty by first using the quadratic formula to compute approximations to the roots and then using a Newton iteration of the type described in Chapter 6 to improve the approximations to the roots.

When $|ac| \ll b^2$, $\sqrt{d} \approx |b|$ and one of the computations $b \pm \sqrt{d}$ suffers from catastrophic cancellation. To eliminate this problem note that

\[
\frac{b + \sqrt{d}}{a} = \frac{c}{b - \sqrt{d}}, \quad \frac{b - \sqrt{d}}{a} = \frac{c}{b + \sqrt{d}}
\]
So, when \( b > 0 \) use
\[
x_+ = \frac{b + \sqrt{d}}{a}, \quad x_- = \frac{c}{b + \sqrt{d}}
\]
and when \( b < 0 \) use
\[
x_+ = \frac{c}{b - \sqrt{d}}, \quad x_- = \frac{b - \sqrt{d}}{a}
\]

**Problem 2.3.15.** Show that the coefficients \( a = 2049, \ b = 4097, \ c = 8192 \) are SP numbers. Use SP arithmetic to compute the roots of the quadratic equation \( ax^2 - 2bx + c = 0 \) using the quadratic formula
\[
x_\pm = \frac{b \pm \sqrt{d}}{a}
\]

**Problem 2.3.16.** Show that the coefficients \( a = 1, \ b = 4096, \ c = 1 \) are SP numbers. Use SP arithmetic to compute the roots of the quadratic equation \( ax^2 - 2bx + c = 0 \) using the quadratic formula.
CHAPTER 2. NUMERICAL COMPUTING

2.4 Matlab Notes

MATLAB provides users with a variety of data types, including integers and floating-point numbers. By default, all floating point variables and constants, and the associated computations, are done in double precision. However, the most recent versions of MATLAB (version 7.0 and higher) allows for the creation of single precision variables and constants. In addition, MATLAB defines certain parameters (such as machine epsilon) associated with floating point computations. In addition to discussing these issues in this section, we also provide some implementation details and several exercises associated with the examples discussed in Section 2.3.

2.4.1 Single Precision Computations in Matlab

By default, MATLAB assumes all floating point variables and constants, and the associated computations, are done in double precision. In order to perform computations using single precision arithmetic, variables and constants must first be converted using the `single` function. Computations involving a mix of SP and DP variables generally produce SP results. For example,

\[
\begin{align*}
\text{theta1} &= 5 * \text{single(pi)}/6 \\
\text{s1} &= \sin(\text{theta1})
\end{align*}
\]

produces the SP values \(\text{theta1} = 2.6179941\) and \(\text{s1} = 0.499998\). That is, because we specify \text{single(pi)}, the constants 5 and 6 in the computation of \text{theta1} are assumed SP, and the computations are done using SP arithmetic.

As a comparison, if we do not specify \text{single} for any of the variables or constants, such as

\[
\begin{align*}
\text{theta2} &= 5 * \text{pi}/6 \\
\text{s2} &= \sin(\text{theta2})
\end{align*}
\]

then MATLAB produces the DP values \(\text{theta2} = 2.61799387799149\) and \(\text{s2} = 0.500000000000000\). However, if the computations are done as

\[
\begin{align*}
\text{theta3} &= \text{single}(5 * \text{pi}/6) \\
\text{s3} &= \sin(\text{theta3})
\end{align*}
\]

then MATLAB produces the values \(\text{theta3} = 2.6179938\), and \(\text{s3} = 0.5000001\). In this case the computation \(5 * \text{pi}/6\) is first done using default DP arithmetic, then the result is converted to SP. Observe, this produces a result that is different than what was computed previously by \text{theta1} and \text{s1} in the pure single precision computation.

2.4.2 Special Constants

One nice feature of MATLAB is that many of the parameters discussed in this chapter can be generated very easily. For example \text{eps} is a function that can be used to compute \(\epsilon_{DP}\) and ulp\(_{DP}(y)\). In particular, \text{eps}(1), \text{eps}('double') and \text{eps} all produce the same result, namely \(2^{-52}\). However, if \(y\) is a DP number, then \text{eps}(y) computes ulp\(_{DP}(y)\), which is simply the distance from \(y\) to the next largest (in magnitude) DP number. The largest and smallest positive DP numbers can be found using the functions \text{realmax} and \text{realmin}.

As with DP numbers, the functions \text{eps}, \text{realmax} and \text{realmin} can be used with SP numbers. For example, \text{eps('single')} and \text{eps(scel(1))} produce the same result, namely \(2^{-23}\). Similarly, \text{realmax('single')} and \text{realmax('single')} return, respectively, the largest and smallest SP floating point numbers. As usual, the \text{doc} command can be used to get more detailed information on any of these functions.

MATLAB also defines parameters for \(\infty\) (called \text{inf} or \text{Inf}) and \(\text{NaN}\) (called \text{nan} or \text{NaN}). It is possible to get, and to use, these quantities in computations. For example:

- A computation such as \(1/0\) will produce \text{Inf}.
- A computation such as \(1/\text{Inf}\) will produce 0.
2.4. MATLAB NOTES

- Computations of indeterminate quantities, such as 0*Inf, 0/0 and Inf/Inf will produce NaN.

**Problem 2.4.1.** In MATLAB, suppose we define the anonymous functions:

\[
\begin{align*}
  f &= \Theta(x) \times \div (x \times (x-1)); \\
  g &= \Theta(x) \div (x-1);
\end{align*}
\]

What is computed by \( f(0) \), \( f(\text{eps}) \), \( g(0) \), \( g(\text{eps}) \), \( f(\text{Inf}) \), and \( g(\text{Inf}) \)? Explain these results.

**2.4.3 Examples**

Section 2.3 provided several examples that illustrate some of the difficulties that can arise in scientific computations. Here we provide some MATLAB implementation details and several exercises associated with these examples.

**Floating-Point Numbers in Output**

When displaying output, it is often necessary to round floating point numbers to fit the number of digits to be displayed. For example, consider writing MATLAB code that sets a DP variable \( x \) to the value 1.99 and prints it twice, first using a floating-point format with 1 place after the decimal point and second using a floating-point format with 2 places after the decimal point. MATLAB code for this could be written as follows:

```matlab
x = 1.99;
fprintf('Printing one decimal point produces %3.1f \n', x)
fprintf('Printing two decimal points produces %4.2f \n', x)
```

The first value printed is 2.0 while the second value printed is 1.99. Of course, 2.0 is not the true value of \( x \). The number 2.0 appears because 2.0 represents the true value rounded to the number of digits displayed. If a printout lists the value of a variable \( x \) as precisely 2.0, that is it prints just these digits, then its actual value may be any number in the range \( 1.95 \leq x < 2.05 \).

A similar situation occurs in the MATLAB command window. When MATLAB is started, numbers are displayed on the screen using the default "format" (called short) of 5 digits. For example, if we set a DP variable \( x = 1.99999 \), MATLAB will display the number as

\[
2.0000
\]

More correct digits can be displayed by changing the format. For example, if we execute the MATLAB statement

```matlab
format long
```
then 15 digits will be displayed; that is, the number \( x \) will be displayed as

\[
1.99999000000000
\]

Other formats can be set; see doc format for more information.

**Problem 2.4.2.** Using the default format short, what is displayed in the command window when the following variables are displayed?

```matlab
x = exp(1)
y = single(exp(1))
z = x - y
```

Do the results make sense? What is displayed when using format long?

**Problem 2.4.3.** Read MATLAB’s help documentation on fprintf. In the example MATLAB code given above, why did the first fprintf command contain \( \backslash n \)? What would happen if this is not used?
Problem 2.4.4. Determine the difference between the following fprintf statements:

```matlab
fprintf(’%6.4f \n’,pi)
fprintf(’%8.4f \n’,pi)
fprintf(’%10.4f \n’,pi)
fprintf(’%10.4e \n’,pi)
```

In particular, what is the significance of the numbers 6.4, 8.4, and 10.4, and the letters f and e?

Plotting a Polynomial

Consider the example from Section 2.3.1, where a plot of

\[ p(x) = (1 - x)^{10} = x^{10} - 10x^9 + 45x^8 - 120x^7 + 210x^6 - 252x^5 + 210x^4 - 120x^3 + 45x^2 - 10x + 1 \]

on the interval \([0.99, 1.01]\) is produced using the power series form of \(p(x)\). This can be done in MATLAB using `linspace`, `plot`, and a very useful function called `polyval`, which is used to evaluate polynomials. Specifically, the following MATLAB code used to produce the plot shown in Fig. 2.1:

```matlab
x = linspace(0.99, 1.01, 103);
c = [1, -10, 45, -120, 210, -252, 210, -120, 45, -10, 1];
p = polyval(c, x);
plot(x, p)
xlabel(’x’), ylabel(’p(x)’)
```

Of course, since we know the factored form of \(p(x)\), we can use it to produce an accurate plot:

```matlab
x = linspace(0.99, 1.01, 103);
p = (1 - x).^10;
plot(x, p)
```

However, not all polynomials can be factored so easily, and it may be necessary to work with the power series form.

Problem 2.4.5. Use the code given above to sketch \(y = p(x)\) for values \(x \in [0.99, 1.01]\) using the power form of \(p(x)\). Pay particular attention to the scaling of the \(y\)-axis. What is the largest value of \(y\) that you observe?

Problem 2.4.6. Construct a figure analogous to Fig. 2.1, but using SP arithmetic rather than DP arithmetic to evaluate \(p(x)\). What is the largest value of \(y\) that you observe in this case?

Repeated Square Roots

The following MATLAB code performs the repeated square roots experiment outlined in Section 2.3.2 on a vector of 10 equally spaced values of \(x\) using 100 iterations (we recommend putting the code in a script m-file):

```matlab
x = 1:0.25:3.75;
n = 100
t = x;
for i = 1:n
    t = sqrt(t);
end
```
for i = 1:n
    t = t .* t;
end
disp(' x      t      x-t ')
disp('================================')
for k = 1:10
    disp(sprintf('%7.4f  %7.4f  %7.4f', x(k), t(k), x(k)-t(k)))
end

When you run this code, the disp and sprintf commands are used to display, in the command window, the results shown in Table 2.1. The command sprintf works just like fprintf but prints the result to a MATLAB string rather than to a file or the screen.

Problem 2.4.7. Write a MATLAB script m-file that implements the above procedure. Experiment using several choices for n. Now rewrite the code so that instead of using equally spaced points for x, a set of random values in the interval [1,4] is chosen. (Hint: See the help documentation for the MATLAB function rand.) Again, experiment using several choices for n.

Estimating the Derivative

The following problems require using MATLAB for experiments related to the example given in Section 2.3.3

Problem 2.4.8. Use MATLAB (with its default DP arithmetic), \( f(x) = \sin(x) \) and \( x = 1 \) radian. Note that \( f'(x) = \cos(x) \) and \( f''(x) = -\sin(x) \). Create a three column table with column headers “n”, “\( -\log_{10}(2r) \)”, and “\( -\log_{10}(2R) \)”. Fill the column headed by “n” with the values 1, 2, \( \cdots \), 51. The remaining entries in each row should be filled with the appropriate values computed using \( h = 2^{-n} \). For what value of \( n \) does the forward difference estimate \( \Delta_{or} f(x) \) of the derivative \( f'(x) \) achieve its maximum accuracy, and what is this maximum accuracy?

Problem 2.4.9. In MATLAB, open the help browser, and search for single precision mathematics. This search can be used to find an example of writing m-files for different data types. Use this example to modify the code from problem 2.4.8 so that it can be used for either SP or DP arithmetic.

A Recurrence Relation

The following problems require using MATLAB for experiments related to the example given in Section 2.3.4

Problem 2.4.10. Reproduce the Table 2.2 using MATLAB single precision arithmetic.

Problem 2.4.11. Repeat the above analysis, but use MATLAB with its default DP arithmetic, to compute the sequence \( \{ \hat{V}_j \}_{j=0}^{23} \). Generate a table analogous to Table 2.2 that displays the values of \( j \), \( \hat{V}_j \), and \( \hat{V}_j \). Hint: Assume that \( \epsilon \), the error in the initial value \( \hat{V}_0 \), has a magnitude equal to half a dp in the DP value of \( V_0 \). Using this estimate, show that \( j \epsilon > 1 \) when \( j \geq 19 \).

Problem 2.4.12. Use the MATLAB DP integration software mentioned in the Software Note below to compute the correct values for \( V_j \) to the number of digits shown in Table 2.2

As mentioned at the end of Section 2.3.4, it is important to emphasize that we are not recommending the use of the recurrence as a method for evaluating the integrals \( \hat{V}_j \). These integrals may be almost trivially evaluated using the MATLAB functions quad and quad1; see Chapter 5 for more details.
CHAPTER 2. NUMERICAL COMPUTING

Summing the Exponential Series

The following problems require using MATLAB for experiments related to the example given in Section 2.3.5

**Problem 2.4.13.** Implement the scheme described in Section 2.3.5, Note 1, in MATLAB DP arithmetic to compute $e^{-20.5}$.

**Problem 2.4.14.** Implement the scheme described in Section 2.3.5, Note 2, in MATLAB DP arithmetic to compute $e^{-20.5}$.

Euclidean Length of a Vector

The following problems require using MATLAB for experiments related to the example given in Section 2.3.6

**Problem 2.4.15.** Write a MATLAB DP program that uses the Moler-Morrison algorithm to compute the length of the vector $\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Your code should display the values of both $p$ and $q$ initially as well as at the end of each trip through the for loop. It should also display the ratio of the new value of $q$ to the cube of its previous value. Compare the value of these ratios to the value of $\frac{1}{4(a^2 + b^2)}$.

**Problem 2.4.16.** Write a MATLAB function to compute the roots of the quadratic equation $ax^2 - 2bx + c = 0$ where the coefficients $a$, $b$ and $c$ are SP numbers. As output produce SP values of the roots. Use DP arithmetic to compute the discriminant $d = b^2 - ac$.

Note that MATLAB does not provide software specifically for computing the roots of a quadratic equation. However, it provides a function, called `roots`, for calculating the roots of a general polynomial and this software solves quadratic equations very carefully. See Section 6.6 for more details.