SIAM J. SCI. COMPUT. © 2013 Society for Industrial and Applied Mathematics
Vol. 35, No. 2, pp. A561–A576

FRACTIONAL SIGNIFICANT DIGITS[*]

JORGEN L. NIKOLAJSEN[†]

Abstract. The concept of fractional significant digits is explored and a definition is proposed, which puts the concept on a firm analytical footing. A simple, reliable procedure is derived for calculating the number of fractional significant digits with high accuracy. Examples are presented, which indicate that fractional significant digits have an important role to play in the comparison of iterative methods, such as eigensolvers and root-finders.

Key words. significant digits, error estimation, iterative processes

AMS subject classification. 65G99

DOI. 10.1137/110828435

1. Introduction.

The concept of significant digits is well known, yet there is disagreement among authors regarding how significant digits should be defined. Ralston and Rabinowitz [1] suggested a definition based on the inequality $|z-\tilde{z}| \leq 0.5\times 10^{-t}$, in which $\tilde{z}$ approximates the exact value z to t significant digits if t is the largest nonnegative integer for which the inequality holds. Vehmanen [2] reviewed several definitions and suggested that t should be defined based on either $|z-\tilde{z}|/max(|z|, |\tilde{z}|) \leq 10^{-t}$ or $|z-\tilde{z}|/max(|z|, |\tilde{z}|, 0.1) \leq 10^{-t}$. Higham [3] introduced the following two definitions: $\tilde{z}$ approximates z to t significant digits if (1) "z and $\tilde{z}$ round to the same number to t significant digits" or (2) "$|z-\tilde{z}|$ is less than half a unit in the tth significant digit of z." Higham then went on to demonstrate the shortcomings of both of these definitions. Lakshmikantham and Sen [4] suggested that t should be based on the inequality $|z-\tilde{z}|/|z| < 5\times 10^{-t}$. Currently, it seems that the most common definition is based on the inequality $|z-\tilde{z}|/|z| \leq 5\times 10^{-t}$; see, for example, Burden and Faires [5]. Clearly, a firm definition of significant digits has not yet been established. The situation is well summarized by Cheney and Kincaid [6], who described the concept of significant digits as "elusive."

Much the same can be said about the concept of fractional significant digits (FSDs). Nikolajsen [7] implicitly defined the number of binary FSDs as $s = -\log_2 e_r$, where $e_r = |z-\tilde{z}|/|z|$ is the relative error. Lakshmikantham and Sen [4] defined decimal FSDs analogously as $s = -\log_{10} e_r$. A web based search brought up several sets of university lecture notes using Lakshmikantham and Sen's definition. However, it is easy to show that this does not always produce realistic results. Consider, for example, $z = 1.000\,000$ and $\tilde{z} = 1.000\,001$, which, according to Lakshmikantham and Sen, result in $s = -\log_{10}(|z-\tilde{z}|/|z|) = 6.0$ significant digits. However, a visual comparison of z and $\tilde{z}$ shows that they have almost 7 digits in common. In fact, their 7th digits differ by only 1 in 10 or 0.1, based on which it may be argued that $\tilde{z}$ is 0.1 digit short of having 7 digits in common with z. That would make $s = 7 - 0.1 = 6.9$ rather than $s = 6.0$ as suggested by Lakshmikantham and Sen. This simple argument is used in the following sections as the starting point for the development of a new

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section March 23, 2011; accepted for publication (in revised form) November 21, 2012; published electronically March 5, 2013.
http://www.siam.org/journals/sisc/35-2/82843.html

[†]Faculty of Computing, Engineering and Sciences, Staffordshire University, Stafford ST18 0AD, UK (j.l.nikolajsen@staffs.ac.uk).

definition of FSDs that more accurately reflects the commonality of digits between z and $\tilde{z}$.

In summary, two methods of calculating the number of FSDs are proposed: Method 1 provides the exact number of FSDs, $s_1 = s_1(z, \tilde{z})$, as a function of the exact value z and the approximate value $\tilde{z}$. s_1 turns out not to be fully continuous. Method 2 is therefore derived, providing an approximate but fully continuous function $s_2 = s_2(z, \tilde{z})$, which is better suited to comparing the relative merits of different iterative procedures, such as eigensolvers and root-finders, as illustrated later.

2. Motivation.

This paper was prompted by the apparent lack of a reliable procedure for comparing the accuracy of iterative eigensolvers. The problems encountered are illustrated in the following sections.

2.1. Comparison of eigenvalues.

Consider Examples 2.1(a) and 2.1(b) which show the known eigenvalues $z_1 = 1.000 \cdot 10^0$ and $z_2 = 9.999 \cdot 10^0$ and the corresponding best approximations $\tilde{z}_1 = 1.001 \cdot 10^0$ and $\tilde{z}_2 = 9.998 \cdot 10^0$ returned by a particular eigensolver. e_{a1} and e_{a2} are the corresponding absolute errors and e_{r2} and e_{r2} are the relative errors. For simplicity, floating-point numbers with a significand length of only 4 decimal digits are used. $(s_1)_1$ and $(s_1)_2$ are the number of FSDs of $\tilde{z}_1$ and $\tilde{z}_2$, respectively. The inner s subscript "1" indicates that these FSDs are associated with Method 1 rather than Method 2, both of which will be derived later. $(s_1)_1$ is calculated as in section 1: z_1 and $\tilde{z}_1$ have 3 leading digits in common and their fourth digits differ by 1 in 10 or 0.1, so $\tilde{z}_1$ is 0.1 short of having 4 digits in common with z_1. Therefore, $\tilde{z}_1$ has $(s_1)_1 = 4 - 0.1 = 3.9$ FSDs. $(s_1)_2$ is also equal to 3.9 for the same reason.

Example 2.1
Eigenvalue approximations.

(a) Eigenvalue #1	(b) Eigenvalue #2
$z_1 = 1.000 \cdot 10^0$	$z_2 = 9.999 \cdot 10^0$
$\tilde{z}_1 = \underline{1.001 \cdot 10^0}$	$\tilde{z}_2 = \underline{9.998 \cdot 10^0}$
$e_{a1} = \lvert z_2 - \tilde{z}_1 \rvert = 0.001 \cdot 10^0$	$e_{a2} = \lvert z_2 - \tilde{z}_2 \rvert = 0.001 \cdot 10^0$
$e_{r1} = e_{a1} / \lvert z_1 \rvert = 1.000 \cdot 10^{-3}$	$e_{r2} = e_{a2} / \lvert z_2 \rvert = 1.000 \cdot 10^{-4}$
$(s_1)_1 = 4 - 0.1 = 3.9$	$(s_1)_2 = 4 - 0.1 = 3.9$

Inspection shows that z_1 and $\tilde{z}_1$ are as close as they can be without coinciding, and so are z_2 and $\tilde{z}_2$. That suggests that the eigensolver has done equally well in extracting both eigenvalues. It is confirmed by the absolute errors being equal at $e_{a1} = e_{a2} = 1.000 \cdot 10^{-3}$ and also by the number of FSDs being equal at $(s_1)_1 = (s_1)_2 = 3.9$. But it is contradicted by the relative errors, $e_{r1} = 1.000 \cdot 10^{-3}$ and $e_{r2} = 1.000 \cdot 10^{-4}$, which differ by a factor 10, suggesting that the eigensolver has extracted Eigenvalue #2 10 times more accurately than Eigenvalue #1; this is a suggestion that is difficult to reconcile with the fact that the approximate values $\tilde{z}_1$ and $\tilde{z}_2$ are both adjacent to the "exact" values z_1 and z_2, respectively. e_{r1} and e_{r2} are clearly important indicators of the relative accuracy of the eigenvalues, but they do not reflect the fact that the eigensolver has managed to place both the approximate values right next to the exact values. The e_a values and the s values do reflect that, so at least one of them appears to have a role to play in quantifying the capabilities of the eigensolver.

Example 2.2
Comparison of eigensolvers.

(a) Eigensolver #1	(b) Eigensolver #2
$z = 1.000 \cdot 10^0$	$z = 1.000 \cdot 10^0$
$\tilde{z}_1 = \underline{1.001 \cdot 10^0}$	$\tilde{z}_2 = \underline{9.999 \cdot 10^{-1}}$
$e_{a1} = \|z - \tilde{z}_1\| = 0.001 \cdot 10^0$	$e_{a2} = \|z - \tilde{z}_2\| = 0.001 \cdot 10^{-1}$
$e_{r1} = e_{a1}/\|z\| = 1.000 \cdot 10^{-3}$	$e_{r2} = e_{a2}/\|z\| = 1.000 \cdot 10^{-4}$
$(s_1)_1 = 4 - 0.1 = 3.9$	$(s_1)_2 = 4 - 0.1 = 3.9$
(c) Eigensolver #3	**(d) Eigensolver #4**
$z = 1.000 \cdot 10^0$	$z = 1.000 \cdot 10^0$
$\tilde{z}_3 = \underline{9.990 \cdot 10^{-1}}$	$\tilde{z}_4 = \underline{9.995 \cdot 10^{-1}}$
$e_{a3} = \|z - \tilde{z}_3\| = 0.010 \cdot 10^{-1}$	$e_{a4} = \|z - \tilde{z}_4\| = 0.005 \cdot 10^{-1}$
$e_{r3} = e_{a3}/\|z\| = 1.000 \cdot 10^{-3}$	$e_{r4} = e_{a4}/\|z\| = 5.000 \cdot 10^{-4}$
$(s_1)_3 = 3 - 0.1 = 2.9$	$(s_1)_4 = 4 - 0.5 = 3.5$

2.2. Comparison of eigensolvers. Disagreements between e_a, e_r, and s also occur in comparisons of different eigensolvers: Examples 2.2(a) and 2.2(b) show the known eigenvalue $z = 1.000 \cdot 10^0$ being approximated by two different eigensolvers. Eigensolvers #1 and #2 have returned best approximations of $\tilde{z}_1 = 1.001 \cdot 10^0$ and $\tilde{z}_2 = 9.999 \cdot 10^{-1}$, respectively. The e_a and e_r values and $(s_1)_1$ are calculated as before. $(s_1)_2 = 4 - 0.1 = 3.9$ follows from the fact that only 1 unit in 10 needs to be added to the 4th digit of $\tilde{z}_2$ to make it equal to z. (A more rigorous method of calculating s in a case like this, where z and $\tilde{z}$ have no actual digits in common, will be presented in section 3.4.)

As in Example 2.1, an inspection shows that $\tilde{z}_1$ and $\tilde{z}_2$ are both as close to z as they can be without coinciding with z, which suggests that Eigensolvers #1 and #2 have done equally well in extracting z. That is also reflected by the number of FSDs being equal at $(s_1)_1 = (s_1)_2 = 3.9$. But it is contradicted by both the absolute errors ($e_{a1} = 1 \cdot 10^{-3}$ and $e_{a2} = 1 \cdot 10^{-4}$) and the relative errors ($e_{r2} = 1 \cdot 10^{-3}$ and $e_{r2} = 1 \cdot 10^{-4}$), which suggest that Eigensolver #2 has done 10 times better than Eigensolver #1. As in Example 2.1, this is difficult to reconcile with the fact that the approximate values $\tilde{z}_1$ and $\tilde{z}_2$, are both adjacent to the "exact" value z. But in contrast to Example 2.1, e_a now agrees with e_r, leaving s as the only parameter indicating that the two eigensolvers did equally well.

The disagreement also shows up when Eigensolvers #1 and #3 are being compared; see Examples 2.2(a) and 2.2(c). Eigenvalue $z = 1.000 \cdot 10^0$ is again being approximated, but now the absolute errors ($e_{a1} = e_{a3} = 1 \cdot 10^{-3}$) and the relative errors ($e_{r1} = e_{r3} = 1 \cdot 10^{-3}$) suggest that Eigensolvers #1 and #3 have done equally well, whereas the number of FSDs ($(s_1)_1 = 3.9$ and $(s_1)_3 = 2.9$) suggest that Eigensolver #1 has done significantly better.

These disagreements are due to the limitations of floating-point numbers: With the four-digit floating-point representation used here, the distance between adjacent significand values is $\Delta S = 0.001$. So in decade 0.1-to-1, where the exponent is $E = -1$, the distance between adjacent floating-point numbers is $\delta_{[0.1:1]} = \Delta S \cdot 10^E = 1.000 \cdot 10^{-4}$, whereas in decade 1-to-10, where the exponent is E = 0, the distance between adjacent floating-point numbers is $\delta_{[1:10]} = \Delta S \cdot 10^E = 1.000 \cdot 10^{-3}$. In Example 2.2(a), $\tilde{z}_1$ resides in decade 1-to-10, so the distance $e_{a1} = 1.000 \cdot 10^{-3}$ between z and $\tilde{z}_1$ is equal to the distance $\delta_{[1:10]} = 1.000 \cdot 10^{-3}$ between adjacent floating-point numbers. In other words, Eigensolver #1 has left a gap of only one floating-point increment between z and $\tilde{z}_1$. That is reflected by the large s value of $(s_1)_1 = 3.9$.

Similarly, in Example 2.2(b), $\tilde{z}_2$ resides in decade 0.1-to-1, so the distance $e_{a2} = 1.000 \cdot 10^{-4}$ between z and $\tilde{z}_2$ is equal to the distance $\delta_{[0.1:1]} = 1.000 \cdot 10^{-4}$ between adjacent floating-point numbers. So Eigensolver #2 has also left a gap of only one floating-point increment between z and $\tilde{z}_2$, as reflected by the large s value of $(s_1)_2 = 3.9$. In other words, both Eigensolvers #1 and #2 have found z with an accuracy that matches the resolution of the floating-point numbers of the decades in which $\tilde{z}_1$ and $\tilde{z}_2$ reside. That suggests that Eigensolvers #1 and #2 have done equally well. But the actual distance between z and $\tilde{z}_1$ is 10 times the distance between z and $\tilde{z}_2$ (as reflected by e_a and e_r) which suggests that Eigensolver #2 did 10 times better than Eigensolver #1.

In Example 2.2(c), $\tilde{z}_3$ resides in decade 0.1-to-1, so the distance $e_{a3} = 1.000 \cdot 10^{-3}$ between z and $\tilde{z}_3$ is 10 times the distance $\delta_{[0.1:1]} = 1.000 \cdot 10^{-4}$ between adjacent floating-point numbers. Thus, Eigensolver #3 has only managed to place $\tilde{z}_3$ within a distance of 10 floating-point increments from z. As already discussed, Eigensolver #1 has managed to place $\tilde{z}_1$ one floating-point increment from z, so in that respect, Eigensolver #1 has done 10 times better than Eigensolver #3. But the actual distance between z and $\tilde{z}_1$ is equal to the distance between z and $\tilde{z}_3$ (as reflected by e_a and e_r) which suggests that Eigensolvers #1 and #3 did equally well.

One reservation regarding the latter suggestion is that the high resolution of $\tilde{z}_3$'s decade (0.1-to-1), compared with the low resolution of $\tilde{z}_1$'s decade (1-to-10), gives Eigensolver #3 an unfair advantage over Eigensolver #1 in demonstrating its capabilities. If the resolution of decade 1-to-10 permitted, Eigensolver #1 might be able to further reduce the gap between $\tilde{z}_1$ and z, thus making $\tilde{z}_1$ more accurate than $\tilde{z}_3$, as measured in terms of the absolute and relative errors.

A comparison of Examples 2.2(a) and 2.2(d) further complicates the picture: Here, the s values, $(s_1)_1$ and $(s_1)_4$, indicate that Eigensolver #1 did better, whereas the e_a and e_r values indicate that Eigensolver #4 did better.

A resolution of this problem is beyond the scope of this paper. Rather, one of the purposes of the paper is to facilitate further investigation by providing a formulation of s that has the same high accuracy as the standard formulations of e_a and e_r.

2.3. Iterative root-finders. The high-accuracy FSD formulation, derived in this paper, can also be used to improve the stopping criteria for iterative root-finders such as, for example, ZANLY in the IMSL Math Library [8]. Such root-finders often declare a root based on the relative error e_r between successive iterates being less than a given error bound. But as shown in the previous sections, an approach based solely on e_r is suspect. If the stopping criteria were instead formulated in terms of maximizing s, then the iterations would continue until the number of floating-point increments between successive iterates were minimized. Thus, the best possible approximation would be returned, regardless of the decade in which it resides. Stopping criteria of this type already exist, see Nikolajsen [7], but a high-accuracy formulation of s is needed to improve their implementation.

Note that when the exact roots are unknown, as in the previous paragraph, s is no longer the number of FSDs but rather the number of matching leading digits of successive iterates. But, for the sake of simplicity, in this paper, s will be referred to as the number of FSDs, which implies that the exact solution is available.

Finally, a precise formulation of the number of FSDs, as provided herein, will serve to put the concept of significant digits on a firm analytical footing, thus, eliminating the "elusiveness" referred to by Cheney and Kincaid [6].

3. Method 1.

A review of section 2 will confirm that the number of FSDs was found using the formula

$$s = s_i + 1 - s_f, \tag{3.1}$$

where s_i is the number of leading digits that z and $\tilde{z}$ have in common and s_f is a fraction that depends on e_a. Equation (3.1) is adopted as a preliminary definition of the number of FSDs. Equation (3.1) is used in this section as the starting point for the development of a formal procedure, called Method 1, for calculating the number of FSDs. For simplicity, a floating-point significand of only 4 decimal digits is used here, but Method 1 (and Method 2 in section 6) work for any significand length.

Example 3.1 shows the basic steps taken to arrive at Method 1. In Example 3.1(a), the number of FSDs is calculated using exactly the same approach as in section 2: $(\tilde{s}_i)_1 = 3$ is the integer number of leading digits that z and $\tilde{z}$ have in common, and $(\tilde{s}_f)_1 = 0.9$ is e_a's significand with the decimal point moved to the position where the string of matching leading digits of z and $\tilde{z}$ ends and the string of nonmatching trailing digits begins. Substitution into (3.1) then gives $\tilde{s}_1 = 3.1$ as shown.

Example 3.1(b) is identical to Example 3.1(a) except that z and $\tilde{z}$'s significands have both been decremented by the smallest amount possible, 0.001. The same $\tilde{s}_1$ value as in Example 3.1(a) is therefore expected but, instead, $\tilde{s}_1$ drops to 2.91 as shown. However, the expected result of $s_1 = 3.1$ can be had by allowing s_1 to be calculated by comparing not only z and $\tilde{z}$ but also z and $\bar{z} = z + e_a$, and $\tilde{z}$ and $\hat{z} = \tilde{z} - e_a$. This results in three s values, $\tilde{s}_1$, $\bar{s}_1$, and $\hat{s}_1$, as shown. The rationale for allowing the comparison between z and $\bar{z}$ is that $\bar{z}$ and $\tilde{z}$ are located on opposite sides of z and are equidistant from z, so z and $\bar{z}$ might have the same number of digits in common as z and $\tilde{z}$. Similarly, $\hat{z}$ and z are located on either side of $\tilde{z}$ and are equidistant from $\tilde{z}$, so $\tilde{z}$ and $\hat{z}$ might also have the same number of digits in common as z and $\tilde{z}$. In Example 3.1(b), $\hat{z}$ turns out to have the expected number of FSDs relative to $\tilde{z}$, i.e., $\hat{s}_1 = 3.1$. Taking s_1 as the maximum of $\tilde{s}_1$, $\bar{s}_1$, and $\hat{s}_1$, as shown, then gives the expected s_1 value of 3.1.

In Example 3.1(c), z and $\tilde{z}$'s significands have again both been decremented by the smallest amount possible, 0.001, so an s_1 value of 3.1 is again expected. Instead, an s_1 value of 2.91 is calculated, using the procedure from Example 3.1(b). One way of obtaining the expected result of $s_1 = 3.1$ is to extend the procedure from Example 3.1(b) to allow z and $\tilde{z}$ to be replaced by any two numbers that are separated by the distance $e_a = |z - \tilde{z}|$. This leads to the following definition of the number of FSDs.

DEFINITION 3.1. *The number of FSDs of $\tilde{z}$ relative to z is the maximum number of leading digits that any two numbers $\tilde{z}_1$ and z_1 can have in common if they are separated by the distance $e_a = |z - \tilde{z}|$.*

As it stands, Definition 3.1 requires that z and $\tilde{z}$ have the same exponent and that z_1 and $\tilde{z}_1$ be restricted to numbers that have the same exponent as z and $\tilde{z}$. This restriction is relaxed later in section 3.4.

An obvious choice for the value of $\tilde{z}_1$'s significand is 1.000, which positions $\tilde{z}_1$ at the start of its decade, thus maximizing the number of digits that the significands of

Example 3.1
Derivation of Method 1.

(a)

$$\begin{aligned} z &= 1.089 \cdot 10^1 \\ \tilde{z} &= \underline{1.080 \cdot 10^1} \\ e_a = |z - \tilde{z}| &= 0.009 \cdot 10^1 \\ \tilde{s}_1 &= (\tilde{s}_i)_1 + 1 - (\tilde{s}_f)_1 \\ &= 3 + 1 - 0.9 = 3.1 \end{aligned}$$

(b)

$$\begin{aligned} z &= 1.088 \cdot 10^1 \\ \tilde{z} &= \underline{1.079 \cdot 10^1} \\ e_a = z - \tilde{z} &= 0.009 \cdot 10^1 \\ \bar{z} = z + e_a &= 1.097 \cdot 10^1 \\ \hat{z} = \tilde{z} - e_a &= 1.070 \cdot 10^1 \end{aligned}$$

$$\begin{cases} \tilde{s}_1 = 2 + 1 - 0.09 = 2.91 \\ \bar{s}_1 = 2 + 1 - 0.09 = 2.91 \\ \hat{s}_1 = 3 + 1 - 0.90 = 3.1 \end{cases}$$

$$s_1 = \max(\tilde{s}_1, \bar{s}_1, \hat{s}_1) = 3.1$$

(c)

$$\begin{aligned} z &= 1.087 \cdot 10^1 \\ \tilde{z} &= \underline{1.078 \cdot 10^1} \\ e_a = z - \tilde{z} &= 0.009 \cdot 10^1 \\ \bar{z} = z + e_a &= 1.096 \cdot 10^1 \\ \hat{z} = \tilde{z} - e_a &= 1.069 \cdot 10^1 \end{aligned}$$

$$\begin{cases} \tilde{s}_1 = 2 + 1 - 0.09 = 2.91 \\ \bar{s}_1 = 2 + 1 - 0.09 = 2.91 \\ \hat{s}_1 = 2 + 1 - 0.09 = 2.91 \end{cases}$$

$$\begin{aligned} s_1 &= \max(\tilde{s}_1, \hat{s}_1, \bar{s}_1) = 2.91 \\ z_1 = \tilde{z}_1 + e_a &= 1.009 \cdot 10^1 \\ \tilde{z}_1 &= \underline{1.000 \cdot 10^1} \\ e_a &= 0.009 \cdot 10^1 \\ s_1 &= 3 + 1 - 0.9 = 3.1 \end{aligned}$$

$\tilde{z}_1$ and $z_1 = \tilde{z}_1 + e_a$ have in common. Thus, in Example 3.1(c), the expected number of FSDs can be found by setting $\tilde{z}_1 = 1.000 \cdot 10^1$ and $z_1 = \tilde{z}_1 + e_a = 1.009 \cdot 10^1$ as shown, resulting in $s_1 = 3.1$.

Several other methods of calculating s were investigated but none were found that matched the accuracy and simplicity of Method 1.

3.1. Binary FSDs. For simplicity, Method 1 was introduced in the previous section using decimal numbers. However, most computers use binary numbers, with decimal numbers being convenient approximations. As a result, decimal FSDs are unreliable in computer applications and must be replaced by binary FSDs, as demonstrated in this section.

Consider Example 3.2 which shows in sequence the smallest numbers that can be accommodated by a 14-bit significand. (A trailing "b" designates a binary number.) Also shown are the corresponding 5- and 4-digit decimal significand values. Notice that 1.0003, 1.0008, and 1.0014 are missing from the 5-digit decimal significand values. Therefore, a 14-bit significand is unable to fully represent a 5-digit decimal significand. On the other hand, the 4-digit decimal significand is overrepresented with several 14-bit numbers representing the same 4-digit decimal number. For example, the entire range of 14-bit binary numbers from entry #6 to entry #13 represent the decimal number 1.001. In other words, the mapping between the 14-bit significand values and 4-decimal-digit significand values is not injective. As a result, the mapping between binary FSDs and decimal FSDs is not injective either.

This is illustrated in Example 3.3. Examples 3.3(a) and (b) take their z and $\tilde{z}$ values from entries #14 and #13 in Example 3.2. Examples 3.3(c) and (d) use entries #13 and #6. Both the binary number of FSDs, $(s_1)_b$, and the decimal number

Example 3.2
Binary numbers and their decimal approximations.

#	14-bit significand	5-dig dec	4-dig dec
1	1.0000 0000 00000b	1.0000	1.000
2	1.0000 0000 00001b	1.0001	1.000
3	1.0000 0000 00010b	1.0002	1.000
4	1.0000 0000 00011b	1.0004	1.000
5	1.0000 0000 00100b	1.0005	1.000
6	1.0000 0000 00101b	1.0006	1.001
7	1.0000 0000 00110b	1.0007	1.001
8	1.0000 0000 00111b	1.0009	1.001
9	1.0000 0000 01000b	1.0010	1.001
10	1.0000 0000 01001b	1.0011	1.001
11	1.0000 0000 01010b	1.0012	1.001
12	1.0000 0000 01011b	1.0013	1.001
13	1.0000 0000 01100b	1.0015	1.001
14	1.0000 0000 01101b	1.0016	1.002
- -	- - -	- - -	- - -

of FSDs, $(s_1)_d$, are calculated in accordance with Method 1. This includes replacing z and $\tilde{z}$ with z_1 and $\tilde{z}_1$, as outlined in the discussion of Example 3.1(c).

The relationship between decimal and binary FSD is governed by $10^{s_d} = 2^{s_b}$ or $s_d = s_b \log_{10} 2$. So, in Example 3.3(a), $(s_1)_b = 13.5$ translates into $(s_1)_d = (s_1)_b \log_{10} 2 = 4.1$, which is not the result found in Example 3.3(b). In fact, $(s_1)_d = 4.1$ is an impossibly large number considering that Example 3.3(b) uses a 4-digit significand. In Examples 3.3(c) and (d), the agreement is just as poor with $(s_1)_b = 11.125$ in Example 3.3(c) translating into $(s_1)_d = (s_1)_b \log_{10} 2 = 3.3$, whereas, according to Example 3.3(d), $(s_1)_d = 4$. Not even the trend is consistent, with $(s_1)_d$ being smaller in Example 3.3(b) than in Example 3.3(d) while $(s_1)_b$ is larger in Example 3.3(a) than in Example 3.3(c). Since the binary numbers are the "exact" numbers and the decimal numbers are approximations, decimal FSDs must be discarded in favor of binary FSDs in computer applications.

3.2. Amendment of (3.1). There is a fundamental problem with $s_1 = 13.5$ in Example 3.3(a): Inspection clearly shows that z_1 and $\tilde{z}_1$ have exactly 13 bits in common. In accordance with Method 1, z and $\tilde{z}$ therefore also have exactly 13 bits in common, so s_1 should be equal to 13.0. This result can be had by replacing (3.1) by

$$s = s_i + not(s_f), \tag{3.2}$$

where the real function $not()$ returns the complement of its binary fraction argument (here s_f) except that any zeros left of the radix point are left unchanged. Thus, in Example 3.3(a), $s_b = 13 + 1 - .1b = 13.5$ should be replaced by $s_b = 13 + not(.1b) = 13 + .0b = 13.0$, which is the correct value according to inspection. Similarly, in Example 3.3(c), inspection shows that z_1 and $\tilde{z}_1$ (and therefore z and $\tilde{z}$) have exactly 11 bits in common, which can be gotten by replacing $(s_1)_b = 11 + 1 - .111b = 11.125$

Example 3.3
Discrepancy between binary and decimal FSDs.

(a)		(b)	
z	$= 1.0000\ 0000\ 01101b$	z	$= 1.002$
$\tilde{z}$	$= \underline{1.0000\ 0000\ 01100b}$	$\tilde{z}$	$= \underline{1.001}$
e_a	$= 0.0000\ 0000\ 00001b$	e_a	$= 0.001$
z_1	$= 1.0000\ 0000\ 00001b$	z_1	$= 1.001$
$\tilde{z}_1$	$= 1.0000\ 0000\ 00000b$	$\tilde{z}_1$	$= 1.000$
$(s_1)_b$	$= 13 + 1 - .1b = 13.5$	$(s_1)_d$	$= 3 + 1 - 0.1 = 3.9$
(c)		(d)	
z	$= 1.0000\ 0000\ 01100b$	z	$= 1.001$
$\tilde{z}$	$= \underline{1.0000\ 0000\ 00101b}$	$\tilde{z}$	$= \underline{1.001}$
e_a	$= 0.0000\ 0000\ 00111b$	e_a	$= 0.000$
z_1	$= 1.0000\ 0000\ 00111b$	z_1	$= 1.000$
$\tilde{z}_1$	$= 1.0000\ 0000\ 00000b$	$\tilde{z}_1$	$= 1.000$
$(s_1)_b$	$= 11 + 1 - .111b = 11.125$	$(s_1)_d$	$= 4$

by $(s_1)_b = 11 + not(.111b) = 11.000$. Experimentation shows that the difference between $not(s_f)$ in (3.2) and $1 - s_f$ in (3.1) is relatively small, decreasing from 0.5 when $s_i = 13$ to near zero when $s_i = 1$.

To further explore the accuracy of (3.2), consider again a significand with a bit length of 14. If $(s_1)_f = .11b$, all but the two rightmost bit pairs of z_1 and $\tilde{z}_1$ match, so s_1 should be equal to 12. That is confirmed by (3.2), which gives s_1 as the sum of $(s_1)_i = 12$ and $not[(s_1)_f] = .00b = .0$. If $(s_1)_f = .10b$, then there is a match between z_1 and $\tilde{z}_1$'s least significant bit pair but a mismatch between the adjacent bit pair, which is twice as significant. Thus, s_1 should be twice as close to 12 as to 13. That is correctly reflected by $not[(s_1)_f] = .01b = 0.25$, leading to $s_1 = 12.25$. This line of argument can be continued for $(s_1)_f$ values with a bit length of 3, in which case $(s_1)_i = 14-3 = 11$ and the possible $(s_1)_f$ values are $(s_1)_f = .111b$, $.110b$, $.101b$, $.100b$, resulting in $not[(s_1)_f] = .000b$, $.001b$, $.010b$, $.011b$. (3.2) then gives the following results: $s_1 = 11.000$, 11.125, 11.250, 11.375. Examination of these values suggests that the results provided by (3.2) again reasonably reflect both the number and the position of the mismatching bit pairs of z_1 and $\tilde{z}_1$. Similar arguments hold for $(s_1)_f$ values with larger bit lengths, confirming that (3.2) provides good accuracy results across the entire range of s values from 1 to l_m, where l_m is the bit length of the significand used.

Note that combinations such as $(s_1)_f = .01b$, $.011b$, $.010b$, etc., cannot occur because, according to Method 1, the radix point must always be inserted next to the leftmost 1-bit of e_a. The consequence is that $(s_1)_f$ can never be less than .1b = 0.5, so $not[(s_1)_f]$ will always be less than 0.5, i.e., $not[(s_1)_f]$ will always be in the interval [0, 0.5) and can never enter the interval [0.5, 1]. Thus, a steady growth in the number of mismatching bit pairs of z_1 and $\tilde{z}_1$ is not reflected in a steady growth of s_1. In other words, function $s_1(z, \tilde{z})$ is not continuous. The discontinuity of $s_1(z, \tilde{z})$ is investigated in the next section.

Example 3.4
Discontinuity of $s_1(z,\tilde{z})$.

(a)	(b)
$z_1 = 1.0000\ 0000\ 11110b$	$z_1 = 1.0000\ 0000\ 11111b$
$\tilde{z}_1 = \underline{1.0000\ 0000\ 00000b}$	$\tilde{z}_1 = \underline{1.0000\ 0000\ 00000b}$
$e_a = 0.0000\ 0000\ 11110b$	$e_a = 0.0000\ 0000\ 11111b$
$s_1 = 9 + not(.11110b) = 9.0313$	$s_1 = 9 + not(.11111b) = 9.0000$

(c)
$z_1 = 1.0000\ 0001\ 00000b$
$\tilde{z}_1 = \underline{1.0000\ 0000\ 00000b}$
$e_a = 0.0000\ 0001\ 00000b$
$s_1 = 8 + not(.1\ 00000b) = 8.4844$

3.3. Continuity. The lack of continuity of $s_1(z,\tilde{z})$ is illustrated in Example 3.4, where z and $\tilde{z}$ have already been shifted to $\tilde{z}_1 = 1.0000\ 0000\ 00000b$ and $z_1 = \tilde{z}_1 + e_a$. Also in Example 3.4, e_a grows by the smallest amount possible, 2^{-13}, from Example 3.4(a) through 3.4(b) to 3.4(c). The resulting s_1 values show a steady reduction from $s_1 = 9.0313$ in Example 3.4(a) to $s_1 = 9.0000$ in Example 3.4(b), followed by a large jump to $s_1 = 8.4844$ in Example 3.4(c), confirming the discontinuity of $s_1(z,\tilde{z})$.

The numerical results reported later show that such intermittent discontinuities occur throughout the range of s_1. They do not prevent s_1 from being used to determine which of several iterative procedures yield the largest number of FSDs, but they do make it difficult to judge accurately the relative merits of iterative procedures when their s_1 values straddle a discontinuity. This difficulty is addressed later by Method 2, which provides an approximate but continuous FSD function $s_2(z,\tilde{z})$.

3.4. z and z̃ in adjacent octaves. In the examples shown so far, z and $\tilde{z}$ reside in the same octave, i.e., they have the same exponent. If they reside in different octaves, and the octaves are not adjacent, then z and $\tilde{z}$ are so widely spaced that they have no leading bits in common, resulting in $s_1 = 0$. If they reside in adjacent octaves, i.e., if the difference between their exponents is one, then s_1 may well be nonzero and large. In order to find s_1 in such cases, an adjustment is needed to e_a as part of the shift from z and $\tilde{z}$ to z_1 and $\tilde{z}_1$.

Consider for example the situation shown in Figure 3.1. $\tilde{z} = 3.0$ resides in octave 2-to-4 and $z = 5.6$ resides in octave 4-to-8. The 14-bit significand used here sets the limit for the number of discrete values that can be accommodated within each octave. In other words, the same number of significand values exists within each octave, regardless of its length. Since octave 4-to-8 is twice as long as octave 2-to-4, the distance between adjacent significand values in octave 4-to-8 is twice as long as in octave 2-to-4. This is indicated symbolically by the distance between the tick marks on the axis in Figure 3.1. Thus, if $\tilde{z}$ and z are shifted up, in accordance with Method 1, to both reside in octave 4-to-8, then the length of $\tilde{e}_a$ (that part of e_a that resides in octave 2-to-4) must be doubled in order for it to represent the same number of significand values as it did in octave 2-to-4. $e_a = |z - \tilde{z}|$ must therefore be replaced by $e_a = 2\tilde{e}_a + \bar{e}_a$ in order to calculate s_1 correctly.

The use of $e_a = 2\tilde{e}_a + \bar{e}_a$ is demonstrated in Example 3.5 for the case shown in Figure 3.1. The example should be self-explanatory. A downshift to octave 2-to-4

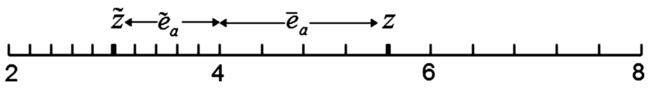

FIG. 3.1. *z and $\tilde{z}$ in different octaves.*

Example 3.5
Upshift to z's octave.

$z = 5.6$	$= 1.0110\ 0110\ 01101b \cdot 2^2$
$\tilde{z} = 3.0$	$= 1.1000\ 0000\ 00000b \cdot 2^1$
Upshift to $\tilde{z}_1$	$= 1.0000\ 0000\ 00000b \cdot 2^2$
$\tilde{e}_a = \tilde{z}_1 - \tilde{z}$	$= 0.0100\ 0000\ 00000b \cdot 2^2$
$\bar{e}_a = z - \tilde{z}_1$	$= \underline{0.0110\ 0110\ 01101b \cdot 2^2}$
$e_a = 2\tilde{e}_a + \bar{e}_a$	$= 0.1110\ 0110\ 01101b \cdot 2^2$
$z_1 = \tilde{z}_1 + e_a$	$= 1.1110\ 0110\ 01101b \cdot 2^2$
$s_1 = 1 + not(.1110\ 0110\ 01101b) = \underline{1.0999}$	

works just as well, requiring $e_a = \tilde{e}_a + \bar{e}_a/2$. It is easy to demonstrate that the result is the same.

4. Implementation of Method 1.

s_1 can be calculated numerically using (3.2) with the following substitutions for $(s_1)_i$ and $not[(s_1)_f]$:

$$(s_1)_i = exponent(\tilde{z}_1) - exponent(e_a), \tag{4.1}$$

$$not[(s_1)_f] = 1 - 2^{(s_1)_i - l_m} - fraction(e_a). \tag{4.2}$$

As before, $\tilde{z}_1$ is the approximation $\tilde{z}$ shifted (as in section 3.4) to the same octave as the "exact" value z. e_a is the absolute error and l_m is the length of the significand, i.e., $l_m = 14$ in the current examples. The intrinsic function *exponent*() returns the binary exponent of its argument plus one (when the argument is written as in the current examples). The intrinsic function *fraction*() returns the significand of its argument with the radix point inserted left of the leftmost 1-bit.

Equations (4.1) and (4.2) will be explained loosely with reference to Example 4.1. Example 4.1 first shows the upshift from z and $\tilde{z}$ to z_1 and $\tilde{z}_1$ and the associated calculation of e_a, as discussed in section 3.4. Next, s_1 is found by inspection, as in the previous examples. Finally, s_1 is found by application of (4.1) and (4.2).

The inspection shows that the correct value of $(s_1)_i$ is 9 because that is the number of matching leading bit pairs of z_1 and $\tilde{z}_1$. $(s_1)_i$ is therefore also equal to the number of leading zeros of the denormalized significand of e_a, i.e., the leftmost e_a significand shown in Example 4.1. If $\tilde{z}_1$'s exponent were equal to zero, then e_a would be equal to its denormalized significand, i.e., $e_a = 0.0000\ 0000\ 11000b = 1.1000b \cdot 2^{-9}$. The correct value of $(s_1)_i$ could then be found as the negative of $exponent(e_a) - 1 = -9$. But in Example 4.1, $\tilde{z}_1$'s nonzero exponent is effectively added to e_a's exponent, resulting in an e_a exponent of $-9 - 8 = -17$, as shown. In order to recover the desired exponent of -9, $\tilde{z}_1$'s exponent ($exponent(\tilde{z}_1) - 1 = -8$) must to be subtracted from the normalized e_a exponent ($exponent(e_a) - 1 = -17$). The result is $exponent(e_a) - 1 - [exponent(\tilde{z}_1) - 1] = -17 - (-8) = -9$, as desired,

Example 4.1
Implementation of Method 1.

$$\begin{aligned}
&z && = 1.0000\ 0000\ 00101b \cdot 2^{-8}\\
&\tilde{z} && = 1.1111\ 1111\ 01101b \cdot 2^{-9}\\
&\tilde{z}_1 && \equiv \underline{1.0000\ 0000\ 00000b \cdot 2^{-8}}\\
&(e_a)_1 = \tilde{z}_1 - \tilde{z} && = 0.0000\ 0000\ 10011b \cdot 2^{-9}\\
&(e_a)_2 = z - \tilde{z}_1 && = \underline{0.0000\ 0000\ 00101b \cdot 2^{-8}}\\
&e_a = 2(e_a)_1 + (e_a)_2 && = 0.0000\ 0000\ 11000b \cdot 2^{-8} = 1.1000 \cdot 2^{-17}\\
&z_1 = \tilde{z}_1 + e_a && = 1.0000\ 0000\ 11000b \cdot 2^{-8}
\end{aligned}$$

<u>Inspection:</u>

$$s_1 = (s_1)_i + (s_1)_f = 9 + \mathit{not}(.11000b) = \underline{9.2188}$$

<u>Method 1:</u>

$$\begin{aligned}
&(s_1)_i = \mathit{exponent}(\tilde{z}_1) - \mathit{exponent}(e_a) = -7 + 16 = 9\\
&(s_1)_f = \mathit{fraction}(e_a) = .11000b\\
&b_1 = 1 - 2^{-l_f} = 1 - 2^{(s_1)_i - l_m} = 1 - 2^{9-14} = .11111b\\
&\mathit{not}[(s_1)_f] = b_1 - (s_1)_f = .11111b - .11000b = .00111b = 0.2188\\
&s_1 = (s_1)_i + \mathit{not}[(s_1)_f] = 9 + 0.2188 = \underline{9.2188}
\end{aligned}$$

the negative of which is $(s_1)_i = \mathit{exponent}(\tilde{z}_1) - \mathit{exponent}(e_a) = 9$, from which (4.1) follows.

Equation (4.2) can also be verified by reference to Example 4.1. The basic definition of $(s_1)_f$ in section 3 makes $(s_1)_f$ equal to $\mathit{fraction}(e_a)$. $\mathit{not}[(s_1)_f]$ (the binary complement of $(s_1)_f$) can then be found by subtracting $(s_1)_f = \mathit{fraction}(e_a)$ from a string of 1-bits, which has the same length as $(s_1)_f$ and is preceded by a radix point. Inspection of Example 4.1 shows that the length of $(s_1)_f$ is $l_f = l_m - (s_1)_i = 14 - 9 = 5$ bits. The bit string from which $(s_1)_f$ should be subtracted must therefore consist of five 1-bits preceded by a radix point, i.e., $b_1 = .11111b$. That can be generated numerically as $b_1 = 1 - 2^{-l_f} = .11111b$. Equation (4.2) then follows from substituting b_1 and $(s_1)_f$ into $\mathit{not}[(s_1)_f] = b_1 - (s_1)_f$. In Example 4.1, $\mathit{not}[(s_1)_f] = b_1 - (s_1)_f = 0.2188$. Substitution into (3.2) then gives $s_1 = 9.2188$, as shown. This agrees with the value found by inspection.

If z or $\tilde{z}$ is zero then the definition of FSDs changes from the number of matching leading bits of z_1 and $\tilde{z}_1$ to the number of leading zeros of $e_a = |z - \tilde{z}|$. This can be found by reducing (4.1) and (4.2) to

$$(s_1)_i = 1 - \mathit{exponent}(e_a), \tag{4.3}$$

$$\mathit{not}[(s_1)_f] = 1 - 2^{1-l_m} - \mathit{fraction}(e_a). \tag{4.4}$$

Equations (4.3) and (4.4) are verified in Example 4.2, which should be self-explanatory.

Application of (4.3) and (4.4) can lead to s_1 values beyond the normal range of 0 to l_m. Thus, s_1 must be zeroed if it becomes less than zero and s_1 must be set equal to l_m if it exceeds l_m.

The implementation of Method 1, as proposed in this section, works regardless of whether z and $\tilde{z}$ are positive or negative and regardless of whether $\tilde{z}$ is larger than

Example 4.2
Method 1 with $z = 0$.

$$\boxed{\begin{array}{ll} z & = 0.0000\ 0000\ 00000b \\ \tilde{z} & = \underline{-1.0101\ 0101\ 01010b \cdot 2^{-8}} \\ e_a & = 1.0101\ 0101\ 01010b \cdot 2^{-8} \\ & = 0.0000\ 0001\ 0101\ 0101\ 01010b \\ \multicolumn{2}{l}{\underline{\text{Inspection:}}} \\ \multicolumn{2}{l}{s = 8 + \mathit{not}(.10101\ 0101\ 01010b) = \underline{8.3333}} \\ \multicolumn{2}{l}{\underline{\text{Method 1:}}} \\ \multicolumn{2}{l}{(s_1)_i = 1 + 7 = 8} \\ \multicolumn{2}{l}{\mathit{not}[(s_1)_f] = 1 - 2^{1-14} - .10101\ 0101\ 01010b = 0.3333} \\ \multicolumn{2}{l}{s_1 = (s_1)_i + \mathit{not}[(s_1)_f] = \underline{8.3333}} \end{array}}$$

or smaller than z. However, (4.1) and (4.2) do not work when z and $\tilde{z}$ have opposite signs, but in that case s_1 will be equal to zero. The reason is that z and $\tilde{z}$ will be entirely uncorrelated when their signs differ because they cannot be in the same octave and they cannot be in adjacent octaves. Even if their exponents are as small as their floating-point representation allows, there will still be an infinity of octaves separating them. On the other hand, e_a will be virtually equal to zero, suggesting that $\tilde{z}$ is an extremely close approximation. But e_r will be of the order of one, suggesting that $\tilde{z}$ is a very poor approximation. This is one more example of the disagreement between e_a, e_r, and s, as discussed in sections 2.1 and 2.2.

Equations (4.1) and (4.2) also do not work and are not needed when $z = \tilde{z}$, in which case $s_1 = l_m$. Finally, as discussed earlier, s_1 must be set equal to zero when $\mathit{exponent}(z_1) \neq \mathit{exponent}(\tilde{z}_1)$, i.e., when z_1 and $\tilde{z}_1$ are not in the same octave.

5. Method 1 results.

Figure 5.1 shows a surface plot of s_1 as a function of z and $\tilde{z}$. z and $\tilde{z}$ are IEEE single-precision numbers with a significand length of $l_m = 24$ bits. z and $\tilde{z}$ both cover the range from 0.875 to 2.5. Thus, Figure 5.1 shows the behavior of $s_1(z, \tilde{z})$ throughout octave 1-to-2 and also in the adjacent quarter octaves. In this section, the designation "octave 1-to-2" refers to the entire square in the $z\tilde{z}$-plane in which $1 \leq z \leq 2$ and $1 \leq \tilde{z} \leq 2$. The smallest possible distance between IEEE single-precision significand values is 2^{-23}. Compared to that, the distance between adjacent points in Figure 5.1 is a crude 2^{-5}, necessitated by the need to cover an entire octave and its vicinity and, at the same time, produce a plot in which the lines are far enough apart so that they do not merge into black, unreadable areas. Spot checks at full resolution (i.e., with 2^{-23}-size increments) have confirmed that no important feature of $s_1(z, \tilde{z})$ has been missed in Figure 5.1.

The top ridge at $s_1 = 24$ coincides with the vertical plane $\tilde{z} = z$ as expected. As z and $\tilde{z}$ diverge, s_1 decreases sharply before leveling out for small values of s_1. A discontinuity, roughly parallel to the $\tilde{z} = z$ plane, occurs each time s_1 passes through an integer value. The discontinuities manifest themselves as steps with a height of about one half unit, as discussed in section 3.2. They are only visible in Figure 5.1 at small values of s_1 because of the low resolution of the plot. The final double-sized step down from $s_1 = 1$ to $s_1 = 0$ is due to the numerical implementation, which sets s_1 equal to zero whenever s_1 is smaller than one. This simplifies the

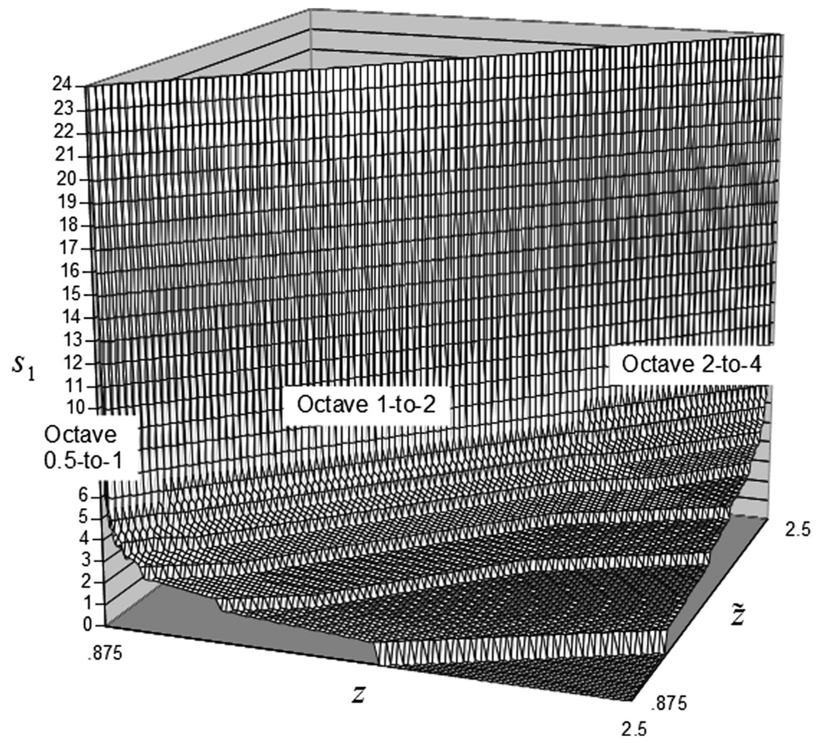

FIG. 5.1. $s_1(z, \tilde{z})$.

implementation significantly and is justified by arguing that two numbers with no leading bits in common can be regarded as uncorrelated for most practical purposes. The discontinuities are discussed further in section 3.3.

Octave 1-to-2 is weakly discernible in the middle of the plot, covering the area outlined by steps that are truly parallel to the vertical plane $\tilde{z} = z$. The bends at the ends of these steps outline the boundaries of octave 1-to-2. The quarter octaves 0.5-to-1 and 2-to-4 cover the areas at the borders of the plot that are also outlined by steps parallel to the $\tilde{z} = z$ plane. Wedged in between are two areas with steps that are not truly parallel to the $\tilde{z} = z$ plane. In these areas, z and $\tilde{z}$ have different exponents. Notice that the quarter octave 0.5-to-1 covers a much smaller area than the quarter octave 2-to-4. This is because the constant increment of 2^{-5} between adjacent points translates into a shorter distance the lower the octave, as discussed in section 3.4. That is also the reason why the plot is steeper the lower the octave.

Only the surface where z is greater than or equal to $\tilde{z}$ is visible in Figure 5.1. The hidden surface is a mirror image of the visible surface, i.e., $s_1(z, \tilde{z})$ is symmetrical about the $\tilde{z} = z$ plane, as expected.

If definition 3.1 in section 3 is accepted, then $s_1(z, \tilde{z})$ is the exact number of FSDs of $\tilde{z}$ relative to z. The obvious shortcoming of s_1 is its lack of continuity, which makes it unsuitable for determining the relative number of FSDs of different iterative methods, as discussed in section 3.3. This problem would disappear if $s(z, \tilde{z})$ could be made continuous, and that is the subject of the next section.

6. Method 2.

Efforts to achieve continuity of $s(z, \tilde{z})$ led to Method 2, which is an approximation of the exact Method 1, in which the terms of (3.2) are calculated as

$$(s_2)_i = exponent(\tilde{z}_1) - exponent(e_a), \tag{6.1}$$

$$not[(s_2)_f] = -\log_2(fraction(e_a) + 2^{(s_2)_i - l_m}). \tag{6.2}$$

The terminology of (6.1) and (6.2) is the same as for (4.1) and (4.2).

Equations (6.1) and (6.2) can be explained loosely as follows, with reference to Example 3.4(b): The integer part of $-\log_2(e_a) = -\log_2(0.0000\ 0000\ 11111b) = 8.0458$ is a rough measure of the number of leading zeros of e_a, i.e., it is a rough measure of $(s_2)_i$. The correct number of leading zeros of e_a, i.e., $(s_2)_i = 9$, can be had by correcting for the effect of $\tilde{z}_1$'s exponent, i.e., $(s_2)_i$ can be found as the integer part of $exponent(\tilde{z}_1) - \log_2(e_a) = 1 + 8.0458 = 9.0458$. The fractional part, $(s_2)_f = 0.0458$, is a measure of how much the actual number of FSDs exceeds 9 so, taken as a whole, s_2 can be found as

$$s_2 = (s_2)_i + (s_2)_f = exponent(\tilde{z}_1) - \log_2(e_a). \tag{6.3}$$

In Example 3.4(b), this gives $s_2 = 9.0458$, but according to Example 3.4(b), the correct fraction is $not[(s_2)_f] = 0.0000$. Experimentation shows that the correct fraction can be had by adding $2^{(s_2)_i - l_m}$ to e_a's significand. Thus, the correction can be made by writing e_a as $e_a = fraction(e_a) \cdot 2^{exponent(e_a)}$ and adding $2^{(s_2)_i - l_m}$ to $fraction(e_a)$. Equations (6.1) and (6.2) then follow from substituting the resulting value of e_a into (6.3). Equation (6.1) gives the formula for $(s_2)_i$ separately to permit substitution of $(s_2)_i$ into (6.2).

For practical purposes, the correction $2^{(s_2)_i - l_m}$ is negligible when s_2 is small, but its significance grows with the value of s_2, as demonstrated by the following extreme case: With the current significand length of $l_m = 14$, the smallest nonzero value of e_a's significand is $0.0000\ 0000\ 00001b = 2^{-13}$ so, by inspection, $s_2 = 13 + not(.1b) = 13$. With $exponent(\tilde{z}_1) = 1$, (3.2) with substitutions from (6.1) and (6.2) also gives $s_2 = 13$, as expected. But (6.3), which omits the correction $2^{(s_2)_i - l_m}$, gives $s_2 = 14$, wrongly suggesting that $z = \tilde{z}$ and confirming the importance of including the correction $2^{(s_2)_i - l_m}$.

In Example 3.4, application of (3.2), with substitutions from (6.1) and (6.2), results in

Example 3.4a: $s_2 = 1 + 8 - \log_2[.11110b + 2^{9-14}] = 9.0458$;

Example 3.4b: $s_2 = 1 + 8 - \log_2[.11111b + 2^{9-14}] = 9.0000$;

Example 3.4c: $s_2 = 1 + 7 - \log_2[.1\ 00000b + 2^{8-14}] = 8.9556$.

The change in s_2 can be seen to be almost constant, illustrating the continuity of the discrete function $s_2(z_1, \tilde{z}_1)$.

If z or $\tilde{z}$ is zero then the definition of FSDs changes from the number of matching leading bits of z_1 and $\tilde{z}_1$ to the number of leading zeros of e_a. This can be found by reducing (6.1) to

$$(s_2)_i = 1 - exponent(e_a) \tag{6.4}$$

while (6.2) remains unchanged. Application of (6.4) and (6.2) can lead to s_2 values beyond the normal range of 0 to l_m. Thus, s_2 must be zeroed if it becomes less than zero and s_2 must be set equal to l_m if it exceeds l_m.

The numerical implementation of Method 2 is subject to the same exceptions as outlined for Method 1 at the end of section 4.

7. Method 2 results.

Figure 7.1 shows a surface plot of s_2 as a function of z and $\tilde{z}$. Figure 7.1 corresponds exactly to Figure 5.1, the only difference being that s_2 is plotted instead of s_1. Figure 7.1 confirms that $s_2(z, \tilde{z})$ is continuous, as expected, except for the step down from $s_2 = 1$ to $s_2 = 0$, which has the same explanation as for s_1 in section 5.

Figure 7.2 shows a sample comparison of s_1 and s_2 with z held constant at $z = 2$ and $\tilde{z}$ traversing the range 1.99994 to 2 in steps of 2^{-23} (i.e., full resolution). The corresponding range of s_1 and s_2 values is 15 to 24 as shown. s_2 is the continuous curve and s_1 is the broken line that touches s_2 at each integer value of s. Additional investigations have confirmed that the close proximity of s_1 and s_2, as seen in Figure 7.2, extends across the entire range of s values from 1 to 24, and throughout all combinations of z and $\tilde{z}$ values. s_2 is therefore considered to be an adequate approximation of s_1 for the purpose of comparing the relative merits of different iterative procedures.

The advantage of using s_2 instead of s_1 for such comparisons is illustrated by the following single-precision example: Assume that two eigensolvers, called a and b, provide the approximations $\tilde{z}_a = 1.937500$ and $\tilde{z}_b = 1.937500 + 2^{-23} \simeq 1.9375001$, respectively, for an ill-conditioned eigenvalue with the 'exact' value $z = 2.000000$. Note that $\tilde{z}_b$ is equal to $\tilde{z}_a$ to within single-precision accuracy ($\sim$7 decimal digits). The two eigensolvers have therefore extracted $z = 2.000000$ to virtually the same accuracy. That is confirmed by the s_2 values being almost equal at $(s_2)_a = 4.999997$ and $(s_2)_b = 5.000000$. But it is contradicted by the s_1 values $(s_1)_a = 4.499990$ and $(s_1)_b = 5.000000$, which wrongly suggest that eigensolver b has extracted the

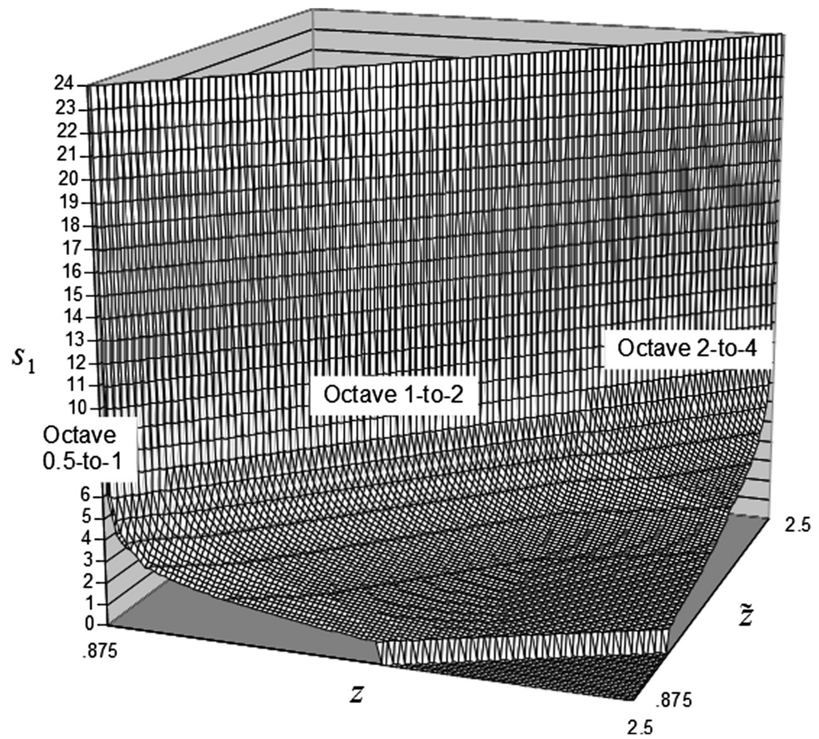

FIG. 7.1. $s_2(z, \tilde{z})$.

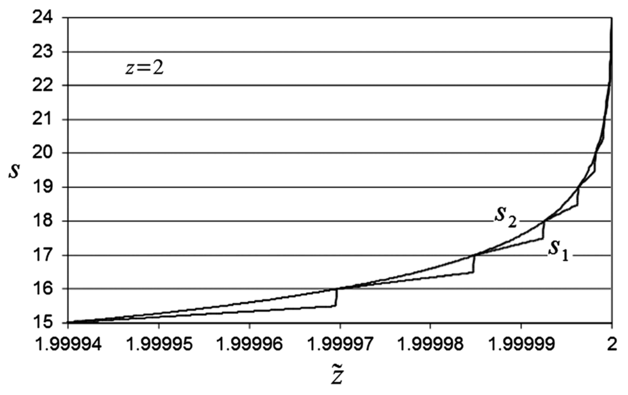

FIG. 7.2. s_1 *and* s_2 *versus* $\tilde{z}$.

eigenvalue with 10% better accuracy than eigensolver a. The discrepancy caused by using s_1 is greatest for small s values. But that makes it no less serious because small s values are associated with ill-conditioned roots, whose successful extraction is of particular concern when evaluating the capabilities of eigensolvers and root-finders.

8. Summary. Examples have been presented which indicate that the number of fractional significant digits (FSDs) cannot be ignored when comparing the relative accuracy of iterative procedures, such as eigensolvers and root-finders.

To facilitate such comparisons, a simple definition has been proposed, which puts the concept of FSDs on a firm analytical footing. It allows the number of FSDs to be determined by a hand calculation, which is demonstrated to be accurate and reliable. With the availability of this simple tool, there seems to be little need for the "elusive" concept of integer significant digits.

Two numerical procedures for calculating the number of FSDs are also presented, providing both the exact number of FSDs and an approximate number of FSDs, the latter being continuous and therefore better suited for comparing the relative merits of different iterative procedures. Both codes are available from the author on request.

REFERENCES

[1] A. Ralston and P. Rabinowitz, *A First Course in Numerical Analysis*, 2nd ed., Dover, Mineola, NY, 1978.
[2] L. Vehmanen, *Significant digits*. Internat. J. Math. Ed. Sci. Tech., 22 (1991), pp. 89–95.
[3] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd Ed., SIAM, Philadelphia, 2002.
[4] V. Lakshmikantham and S. K. Sen, *Computational Error and Complexity in Science and Engineering*, Elsevier, Amsterdam, The Netherlands, 2005.
[5] R. L. Burden and J. D. Faires, *Numerical Analysis*, 8th ed., Thomson Learning, Belmont, CA, 2005.
[6] W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*, 6th ed., Thomson Learning, Belmont, CA, 2008.
[7] J. L. Nikolajsen, *An improved Laguerre eigensolver for unsymmetric matrices*, SIAM J. Sci. Comput., 22 (2000), pp. 822–834.
[8] *IMSL Fortran Numerical Library—User's Guide—Math Library*, Version 7.0, Rogue Wave Software, Boulder, CO, 2010.