IMA Journal of Numerical Analysis (1988) **8**, 517–526

The Symmetric Level-Index System

C. W. Clenshaw and P. R. Turner

Department of Mathematics, University of Lancaster, Lancaster LA1 4YL, UK

Dedicated to Professor Leslie Fox on the occasion of his seventieth birthday

[Received 16 September 1986 and in revised form 4 June 1987]

The purpose of this paper is to present the details of an arithmetic system which virtually abolishes the phenomena of computer overflow and underflow in a logically symmetric manner. A generalized exponential function is used in such a way as to enable very large numbers to be represented with a uniform precision, and very small numbers by reciprocation.

1. Introduction

In this paper we discuss in detail the symmetric version of the level-index (li) number system and its arithmetic. The li system was introduced by Clenshaw & Olver (1984) and the algorithms for li arithmetic were presented by the same authors (1987). This 1987 paper, an essential introduction to the present work, is henceforth referred to as 'AO', the abbreviation signifying 'arithmetic operations'. In both those papers, mention was made of an extension of the system which allows a small number to be represented to high precision by storing its reciprocal in li form, together with an indicator to show that this is indeed a reciprocal.

Thus, for the symmetric level index (sli) system, we may represent a nonzero real number X by another number x together with two signs $\mathrm{s}(X)$ and $\mathrm{r}(X)$. The relation between these quantities is

$$X = \mathrm{s}(X)\phi(x)^{\mathrm{r}(X)}, \tag{1}$$

where $\mathrm{s}(X) = \operatorname{sgn} X$,

$$\mathrm{r}(X) = \begin{cases} +1 & \text{if } |X| \geqslant 1, \\ -1 & \text{if } |X| < 1, \end{cases}$$

and ϕ is the generalized exponential function used by Clenshaw & Olver (1984, 1987) for the ordinary li representation. This is defined for nonnegative arguments by

$$\phi(x) = \begin{cases} x & \text{if } 0 \leqslant x \leqslant 1, \\ \exp \phi(x-1) & \text{if } x > 1. \end{cases}$$

There are several possible implementations of this representation, depending on how the (now redundant) level zero is used. One option is to permit the use of levels 1 to 8 instead of 0 to 7; another to retain level zero for special purposes. There is also redundancy in the representation of ± 1 (since $\phi(1) = 1/\phi(1)$). This could be used for a special representation of zero.

An alternative representation function for sli numbers which was introduced in AO uses the mapping $X = s(X)\Phi(x)$. Here $|X| = \Phi(x)$, where

$$\Phi(x) = \begin{cases} \phi(1+x) & \text{if } x \geqslant 0 \\ 1/\phi(1-x) & \text{if } x < 0 \end{cases}$$

and $s(X)$ is still $\operatorname{sgn} X$. The two representations are essentially similar and we shall use whichever is convenient.

It is obvious that sli arithmetic can be performed merely by using the li algorithms; but this is likely to be unnecessarily time-consuming since, for example, to compute the sum of two small numbers by the relation

$$\frac{1}{\phi(x)} + \frac{1}{\phi(y)} = \frac{\phi(x) + \phi(y)}{\phi(x)\phi(y)}$$

requires three li operations. It is also likely to be less satisfactory because the computation of the sum and product will be based on the *larger* of x and y, that is, on the *less significant* summand.

For these two reasons, we describe (in Section 2) algorithms for sli arithmetic which use the same basic approach as the li operations but work directly with the sli representation; the primary objective being to provide sli algorithms which are as fast to execute as those for li arithmetic. In Section 3, we discuss briefly the first-order error analysis of these algorithms, from which we see that appropriate precision can be preserved in a similar way to that of the li system.

For the sli system to be a useful computing facility, an efficient hardware/software implementation is required. The same considerations apply here as to the li system, for which some approaches to practical implementation are discussed by Turner (1985). It is also necessary that the system be seen to be useful in application. Clenshaw & Turner (1988) give an indication of this usefulness and demonstrate that the appropriate degree of precision is retained in the very large and very small numbers generated, so that useful information is obtained as the end-product. This point is discussed in more detail by Lozier & Olver (1988).

2. Algorithms for sli arithmetic

One of the main advantages of the sli system, compared with li, lies in the fact that the absence of level zero removes many of the special cases. An extra difficulty which is introduced is the occasional need for 'flip-over' between reciprocal and standard form: this happens, for example, when

$$\phi(x) - \phi(y) < 1 \quad \text{or} \quad \frac{1}{\phi(x)} + \frac{1}{\phi(y)} \geqslant 1,$$

so that the reciprocation sign of the answer is different from that of the operands. However, all such cases are easily identified and dealt with.

We begin the discussion of arithmetic algorithms with addition and subtraction; that is, we must find

$$Z = s(Z)\phi(z)^{r(Z)} = X \pm Y = s(X)\phi(x)^{r(X)} \pm s(Y)\phi(y)^{r(Y)}.$$

We assume throughout the rest of this discussion that

$$\mathrm{s}(Z)=\mathrm{s}(X)=\mathrm{s}(Y)=+1 \quad \text{and} \quad X \geqslant Y.$$

(The sorting of signs, including the operation sign, and the ordering of the arguments, is a necessary preliminary step to the algorithms. We do not discuss these details here.) There are therefore three situations to consider, namely:

$$\mathrm{r}(X)=\mathrm{r}(Y)=+1, \qquad \mathrm{r}(X)=+1=-\mathrm{r}(Y), \qquad \mathrm{r}(X)=\mathrm{r}(Y)=-1.$$

These are referred to, respectively, as 'large', 'mixed', and 'small' arithmetic. It should also be noted that in *all* cases we have

$$x,y,z \geqslant 1.$$

For large arithmetic, the algorithms use the sequences $\{a_j\}$, $\{b_j\}$, and $\{c_j\}$ of the original li operations; thus

$$a_j=1/\phi(x-j), \qquad b_j=\phi(y-j)/\phi(x-j), \qquad c_j=\phi(z-j)/\phi(x-j).$$

For mixed and for small arithmetic, we still use $\{a_j\}$ and $\{c_j\}$. However, the sequence $\{b_j\}$ is replaced by $\{\alpha_j\}$ in the mixed case, and $\{\beta_j\}$ in the small case, where

$$\alpha_j=1/\phi(y-j), \qquad \beta_j=\phi(x-j)/\phi(y-j).$$

In some situations, it is also necessary to compute elements of the sequence $\{h_j\}$, where

$$h_j=\phi(z-j),$$

just as in li arithmetic when division is performed with a level-zero denominator, for example.

Throughout the subsequent description, we denote the integer and fractional parts of x by l and f respectively, and those of y by m and g.

Note that $x \geqslant y$ for large arithmetic, while $x \leqslant y$ in the case of small arithmetic. The subtraction of two equal numbers would of course be identified as a special case. We thus assume henceforth that $X>Y$. (The algorithm remains valid for $X+X$, but this too can be treated separately since then either $b_0=1$ or $\beta_0=1$.)

The algorithm for addition or subtraction proceeds as follows:

Step 1. Set $\mathrm{r}(Z):=\mathrm{r}(X)$.

Step 2. Compute

$$a_{l-1}:=\mathrm{e}^{-f}, \qquad a_{j-1}:=\exp(-1/a_j) \quad (j=l-1,l-2,\ldots,1)$$

in parallel with

If $\mathrm{r}(X)=\mathrm{r}(Y)=1$:	if $\mathrm{r}(X)=1$ and $\mathrm{r}(Y)=-1$:	if $\mathrm{r}(X)=\mathrm{r}(Y)=-1$:
$b_{m-1}:=a_{m-1}\mathrm{e}^{g}$,	$\alpha_{m-1}:=\mathrm{e}^{-g}$,	$\beta_{l-1}:=\exp[f-\exp^{(m-l)}g]$,
$b_{j-1}:=\exp[(b_j-1)/a_j]$	$\alpha_{j-1}:=\exp(-1/\alpha_j)$	$\beta_{j-1}:=\exp[(\beta_j-1)/a_j\beta_j]$
$(j=m-1,m-2,\ldots,1)$,	$(j=m-1,m-2,\ldots,1)$,	$(j=l-1,l-2,\ldots,1)$,
$c_0'':=1\pm b_0$;	$c_0'':=1\pm a_0\alpha_0$;	$c_0':=1\pm\beta_0$.

```
Step 3.  If $c_0'' < a_0$ then                          | if $a_0 c_0' > 1$ then set
             $r(Z) := -1$,                               |     $r(Z) := 1$,
             $h_1 := -\ln(c_0''/a_0)$,                   |     $z := 1 + \ln(a_0 c_0')$,
             go to step 5;                               |     finish;
         else  if $l = 1$ then $h_1 := f + \ln c_0''$,   | else  if $l = 1$ then $h_1 := f - \ln c_0'$,
                         go to step 5;                   |                 go to step 5;
               else $c_1 := 1 + a_1 \ln c_0''$;          |       else $c_1 := 1 - a_1 \ln c_0'$;

Step 4.  For $j = 1, \ldots, l-2$:
             if $c_j < a_j$              then $z := j + c_j/a_j$, finish;
                                         else $c_{j+1} := 1 + a_{j+1} \ln c_j$;
           if $c_{l-1} < a_{l-1}$      then $z := l - 1 + c_{l-1}/a_{l-1}$, finish;
                                         else $h_l := f + \ln c_{l-1}$.

Step 5.  Compute $h_j := \ln h_{j-1}$ until $h_j \in [0, 1)$ in which case
             $z := j + h_j$.
```

Notes. (i) In the case of 'small' arithmetic, the computation of β_{l-1} may require (when $m > l$) the computation of

$$\alpha_{m-1} := e^{-g}, \qquad \alpha_{j-1} := \exp(-1/\alpha_j) \quad (j = m-1, \ldots, l),$$

in order to obtain $\beta_{l-1} := \alpha_{l-1}/a_{l-1}$. This still requires just one sequence of length l and one of length m in order to obtain c_0'.

A somewhat simpler relation for the sequence $\{\beta_j\}$ is

$$\beta_{j-1} := \exp[(\beta_j - 1)/\alpha_j],$$

which corresponds precisely to the defining relation for $\{b_j\}$. This necessitates the computation of the full sequence $\{\alpha_j\}$, as well as $\{a_j\}$, first. In an environment where these can be computed in parallel (or if several sli operations are being performed in parallel on a machine such as the ICL DAP), this simpler relation may be preferred. However, greater care is then needed over the precisions of the various quantities, since this has the effect of basing the computation on the smaller operand $1/\phi(y)$ which complicates the error analysis.

(ii) The various cases of 'flip-over' are easily identified in step 3 and handled using (repeated) logarithms.

(iii) In the ordinary li algorithms, a test is made so that, if (the computed value of) a_0 is zero, then $z = x$. Similar provisos are useful here. For mixed arithmetic we set $z = x$ if $\alpha_0 = 0$, while for small arithmetic the condition $\beta_0 = 0$ is used.

(iv) In the cases of large or mixed addition and small subtraction, we know that $c_j \geq a_j$ $(j = 1, \ldots, l-1)$ and so the testing implied by step 4 is unnecessary in these instances. It is included in the algorithm for the sake of compactness.

(v) All the internal calculation can be in fixed-point arithmetic, since the various quantities remain suitably bounded. In all cases, we have

$$0 \leqslant a_j, b_j, \alpha_j, \beta_j \leqslant 1,$$

while $0 \leqslant c_j \leqslant 2$ except possibly for small subtraction. For this exceptional case, we have $c_0' \geqslant \gamma$, where γ is the working precision being used, from which it follows that $c_1 = 1 - a_1 \ln c_0' \leqslant 1 + \ln 1/\gamma$. A similar bound applies for h_l whenever it is computed, while for large or mixed subtraction with $c_0 < a_0$

$$h_1 = \ln a_0/c_0 \leqslant \ln 1/c_0 \leqslant \ln 1/\gamma.$$

(vi) For extended sums, savings can be made in much the same way as in AO(§2.5). Only one sequence $\{a_j\}$ is needed, and any $\{b_j\}$ and $\{\alpha_j\}$ (or $\{\beta_j\}$) can be computed in parallel to produce a single c_0'' (or c_0').

The multiplication and division of sli numbers are easily described, since all the various cases are equivalent to operations which involve only numbers greater than unity together with the appropriate reciprocation signs; specifically:

$$\phi(x)\phi(y) = [\phi(x)^{-1}\phi(y)^{-1}]^{-1} = \phi(x)/\phi(y)^{-1} = [\phi(x)^{-1}/\phi(y)]^{-1},$$
$$\phi(x)/\phi(y) = \phi(x)\phi(y)^{-1} = [\phi(x)^{-1}\phi(y)]^{-1} = [\phi(x)^{-1}/\phi(y)^{-1}]^{-1}.$$

Thus we may suppose $X \geqslant Y \geqslant 1$ and consider just XY and X/Y, which are readily dealt with using the ordinary li algorithms for

$$\ln X \pm \ln Y = \phi(x-1) \pm \phi(y-1)$$

(or, equivalently, by setting $c_1 = 1 \pm b_1$ in the above algorithm with minor modification for the case $m = 1$). Comparison with AO highlights the simplicity which the symmetric representation allows.

The exponentiation operation does, however, require some use of the ordinary li division and multiplication algorithms in situations such as raising a large number to a small power; that is,

$$Z = \phi(z)^{r(Z)} = X^Y = \phi(x)^{\phi(y)^{-1}}.$$

Then $r(Z) = 1$ and

$$\ln Z = \phi(z-1) = \phi(y)^{-1}\phi(x-1),$$

so that, if $y > x - 1$, the sli operation would compute the right-hand side as $[\phi(y)/\phi(x-1)]^{-1}$. What is required is the level-zero result of the li operation $\phi(x-1)/\phi(y) = h$ (say), from which we obtain $z = 1 + h$. It is important to note here that the li division algorithm can be significantly simplified, since $y \geqslant 1$ and so division by a level-zero number is never needed.

3. Error control

In this section we confirm, by means of linearized error analysis, that the errors resulting from the above algorithms can be restricted to the order of inherent error. This is achieved by working to fixed absolute precisions in the computation of the various sequences. The implications of the analysis on the choice of working precisions are discussed briefly in the final section of the paper.

We adopt here the same notation as in AO (§3.2); thus we assume that the sequence $\{a_j\}$ is stored with absolute precision γ_1 while all the other quantities are stored to absolute precision γ. We also assume that the various special functions involved are computed to these same precisions. However, it should be noted that, especially in the case of $\{\beta_j\}$, considerable care will be needed in the implementation. In order to evaluate $\exp[(\beta_j - 1)/a_j\beta_j]$ with absolute precision γ, it will not suffice to compute the product $a_j\beta_j$ to this same precision. Similar considerations for the sequence $\{c_j\}$ led, in AO, to the greater precision requirement for the sequence $\{a_j\}$.

In performing a first-order error analysis for the sli algorithms, there are nine cases to consider, but most of these are straightforward extensions of the li theory in AO. Two of these cases, namely 'large' addition and subtraction where the result is at least unity are of course identical to the li results without the need to consider the special cases arising from level-zero operands or answers.

For subtraction where the result is in reciprocal form, that is, where we have $x - y \geqslant \gamma$ and

$$\phi(x) - \phi(y) = 1/\phi(z),$$

the inherent error given by linear perturbation theory is

$$\delta z = -\frac{\phi'(x)\phi(z)}{\phi'(z-1)}\delta x + \frac{\phi'(y)\phi(z)}{\phi'(z-1)}\delta y$$

where we have used the fact (cf. AO, eqn 3.2) that

$$\frac{\phi(z)^2}{\phi'(z)} = \frac{\phi(z)^2}{\phi(z)\phi'(z-1)} = \frac{\phi(z)}{\phi'(z-1)}.$$

The analysis of this case is precisely the same as that for ordinary li subtraction as far as the computation of $c_0/a_0 = h_0'$(say). (This is the h of AO (eqn 2.15) in the case $n = 0$.) Then, for $j = 0, 1, \ldots,$ we have

$$|\delta h_{j+1}| \leqslant \frac{1}{h_j}|\delta h_j| + \gamma \leqslant \gamma\left(1 + \frac{1}{h_j} + \cdots + \frac{1}{h_j \cdots h_1}\right) + \frac{1}{h_j \cdots h_1 h_0}|\delta h_0'|.$$

Now $1/h_0' = \phi(z)$ and $h_j = \phi(z - j)$ $(j = 1, 2, \ldots)$, so that, using AO (eqn 3.17), we have

$$|\delta h_{j+1}| \leqslant \rho\gamma + \frac{\phi(z)}{\phi'(z-1)}|\delta h_0'| \tag{3.1}$$

$\left(\text{as in AO, } \rho \text{ denotes the sum } 1 + \frac{1}{\phi'(1)} + \frac{1}{\phi'(2)} + \cdots = 2{\cdot}3921\right).$

The first part of the following technical lemma establishes that

$$1 \leqslant \phi(z)/\phi'(z-1)$$

and so the bounds obtained in AO (§3.4) adapted to the special case $n = 0$ yield

$$|\delta z| \leqslant \left[(2\rho + 1)\gamma + \lambda\gamma_1\left(1 + \frac{\rho}{e}\right)\right]\frac{\phi(z)\phi'(x)}{\phi'(z-1)}, \tag{3.2}$$

where λ, as in AO (eqn 3.14), $\lambda = (4+e^2)e^{-e}+1 = 1{\cdot}7515\ldots$.

LEMMA 3.1 (i) *The function* θ, *given by*

$$\theta(z) = \phi(z)/\phi'(z-1),$$

is increasing for $z > 1$, *and* $\theta(1) = 1$.

(ii) *For* $1 \leqslant j \leqslant z-1$ *and* $\phi(z) \geqslant e^2$, *the quantity* $\theta(z-j)/\theta(z)$ *is a decreasing function of* z.

Proof. Now $\theta(1) = \phi(1)/\phi'(0) = 1$ and

$$\theta'(z) = \frac{\phi'(z)}{\phi'(z-1)} - \frac{\phi(z)\phi''(z-1)}{\phi'(z-1)^2} = \phi(z)\left(1 - \frac{\phi''(z-1)}{\phi'(z-1)^2}\right).$$

If $n = \lfloor z \rfloor = 1$, then $\phi''(z-1) = 0$, so that $\theta'(z) > 0$; while, for $n \geqslant 2$, we have

$$\phi''(z-1) = \phi'(z-1)\sum_{j=2}^{n} \phi'(z-j),$$

so that

$$\frac{\phi''(z-1)}{\phi'(z-1)^2} = \frac{\sum_{j=2}^{n}\phi'(z-j)}{\phi'(z-1)} \leqslant \frac{n-1}{\phi(z-1)} \leqslant \frac{n-1}{\phi(n-1)} < 1,$$

which again yields $\theta'(z) > 0$ and completes the proof of (i). Similar arguments can be used to establish (ii). □

The analyses of the 'mixed' operations, with result greater than unity, are again simple modifications of the ordinary li cases, and the only changes result from the use of the $\{\alpha_j\}$ sequence. We obtain for the addition $\phi(x) + 1/\phi(y) = \phi(z)$ the bound

$$|\delta z| \leqslant (\rho + \lambda + 3)\gamma + (\rho \ln 1/\gamma + 1)\lambda\gamma_1, \tag{3.3}$$

while for $\phi(x) - 1/\phi(y) = \phi(z)$ we have

$$|\delta z| \leqslant [(\rho + \lambda + 1)\gamma + (\rho \ln 1/\gamma + 1)\lambda\gamma_1]\phi'(x)/\phi'(z). \tag{3.4}$$

For the case where this subtraction has a result less than unity, we require z such that

$$\phi(x) - 1/\phi(y) = 1/\phi(z).$$

Here the analysis is simplified by the fact that $\phi(x) \in [1, 2)$, so that $l = 1$ and $\phi(x) = \phi'(x)$. This yields $|\delta c_0| \leqslant \gamma_1 + (\lambda+1)\gamma$, from which, using (3.1), we obtain the required bound in the form

$$|\delta z| \leqslant [(\lambda + \rho + 2)\gamma + 2\gamma_1]\frac{\phi(x)\phi(z)}{\phi'(z-1)}. \tag{3.5}$$

In the common cases of 'small' arithmetic where

$$\frac{1}{\phi(x)} \pm \frac{1}{\phi(y)} = \frac{1}{\phi(z)},$$

the inherent error given by linear perturbation theory satisfies

$$\frac{\phi'(z-1)}{\phi(z)}\delta z=\frac{\phi'(x-1)}{\phi(x)}\delta x\pm\frac{\phi'(y-1)}{\phi(y)}\delta y.$$

Here $x\leqslant y$, and so from Lemma 3.1 we see that the first term on the right hand side dominates; that is,

$$|\delta z|\simeq\frac{\phi'(x-1)\phi(z)}{\phi'(z-1)\phi(x)}|\delta x|.$$

The essential distinguishing feature in the analysis of 'small' arithmetic results from analysing the sequence $\{\beta_j\}$.

From this we obtain

$$|\delta\beta_0|\leqslant\frac{\phi'(y-1)}{\phi(y)}\phi(x)\left(\frac{A\lambda\gamma_1}{\mathrm{e}}+B\gamma+K\frac{\phi(y-l+1)}{\phi'(y-l)\phi(x-l+1)}\right),$$

where

$$A=\sum_{j=0}^{l-2}\frac{\phi(y-j)}{\phi'(y-j-1)}\frac{\phi(x-j-1)}{\phi(x-j)},\qquad B=\sum_{j=0}^{l-2}\frac{\phi(y-j)}{\phi'(y-j-1)\phi(x-j)},$$

and $K=\mathrm{e}(\lambda\gamma+\gamma_1)+\gamma$.

Now Lemma 3.1 shows that, for fixed x, the above bound decreases as y increases. Since $y\geqslant x$ and $\lfloor x\rfloor=l$, it follows that $A\leqslant\rho$ and $B\leqslant\rho-1$, and Lemma 3.1 gives

$$\frac{\phi'(y-1)}{\phi(y)}\phi(x)\leqslant\phi'(x-1).$$

Hence

$$\begin{aligned}|\delta\beta_0|&\leqslant\phi'(x-1)\left(\frac{\rho\lambda}{\mathrm{e}}\gamma_1+(\rho-1)\gamma+K\right)\\&=\phi'(x-1)[\gamma(\rho+\mathrm{e}\lambda)+\gamma_1(\mathrm{e}+\rho\lambda/\mathrm{e})]\\&=K_1\phi'(x-1)\quad\text{(say)}.\end{aligned}\tag{3.6}$$

The analysis of the sequence $c_0', c_1, \ldots$ is much the same as that in AO and yields, for subtraction (where now $y-x\geqslant\gamma$), the bound

$$|\delta c_{l-1}|\leqslant\left(1+\frac{a_{l-1}}{c_{l-2}}+\cdots+\frac{a_{l-1}\cdots a_2}{c_{l-2}\cdots c_1}\right)\gamma_2+\frac{a_{l-1}\cdots a_1}{c_{l-2}\cdots c_1c_0'}|\delta\beta_0|,$$

where, again following AO,

$$\gamma_2=\gamma+\lambda\gamma_1\ln 1/\gamma.\tag{3.7}$$

Here $c_j=\phi(z-j)/\phi(x-j)>1$ and so we have, using (3.6),

$$|\delta c_{l-1}|\leqslant\rho\gamma_2+\frac{a_{l-1}\cdots a_1}{c_{l-2}\cdots c_1c_0'}K_1\phi'(x-1)=\rho\gamma_2+\frac{K_1}{c_{l-2}\cdots c_1c_0'}.$$

Now $c_0'=\phi(x)/\phi(z)$ and

$$c_1\cdots c_{l-2}=\frac{\phi(z-1)\cdots\phi(z-l+2)}{\phi(x-1)\cdots\phi(x-l+2)}=\frac{\phi'(z-1)}{\phi'(x-1)}\frac{\phi(x-l+1)}{\phi'(z-l+1)},$$

so that

$$|\delta h_l| \leqslant \frac{1}{c_{l-1}} |\delta c_{l-1}| + \gamma \leqslant \gamma + \rho\gamma_2 + K_1 \frac{\phi'(x-1)\phi(z)}{\phi'(z-1)\phi(x)} \phi'(z-l).$$

Similar arguments to those used above for $h_{l+1}, h_{l+2}, \ldots$ finally yield

$$\begin{aligned} |\delta z| &\leqslant \rho(\gamma+\gamma_2) + K_1 \frac{\phi'(x-1)\phi(z)}{\phi'(z-1)\phi(x)} \\ &\leqslant \frac{\phi'(x-1)\phi(z)}{\phi'(z-1)\phi(x)} \left[\gamma(3\rho + e\lambda) + \gamma_1\left(\rho\lambda \ln\frac{1}{\gamma} + e + \frac{\rho\lambda}{e}\right)\right], \end{aligned} \tag{3.8}$$

which is the required form.

In the case of addition, the same analysis yields, for $j \geqslant 1$,

$$\begin{aligned} |\delta c_j| &\leqslant \left(1 + \frac{a_j}{c_{j-1}} + \cdots + \frac{a_j \cdots a_2}{c_{j-1} \cdots c_1}\right)\gamma_2 + \frac{a_j \cdots a_1}{c_{j-1} \cdots c_1 c_0} |\delta\beta_0| \\ &\leqslant \left(1 + \frac{a_j}{c_{j-1}} + \cdots + \frac{a_j \cdots a_2}{c_{j-1} \cdots c_1}\right)\gamma_2 + K_1 \frac{\phi'(x-1)\phi(z)}{\phi'(z-1)\phi(x)} \frac{\phi'(z-j)}{\phi(x-j)}. \end{aligned}$$

From this we find that, whenever $1/\phi(x) + 1/\phi(y) = 1/\phi(z)$, then the bound

$$|\delta z| \leqslant \left[\gamma(3\rho + e\lambda - 1) + \gamma_1\left(4\lambda + e + \frac{\rho\lambda}{e} + 2(\rho-1)\lambda \ln\frac{1}{\gamma}\right)\right] \frac{\phi'(x-1)\phi(z)}{\phi'(z-1)\phi(x)}. \tag{3.9}$$

is satisfied for both $\lfloor z \rfloor = l$ and $\lfloor z \rfloor < l$.

For the special case of small addition where the result is greater than unity and we seek z such that

$$\phi(z) = 1/\phi(x) + 1/\phi(y),$$

we have $\phi(z), \phi(x) \in [1, 2]$. In this case, we obtain the absolute bound

$$|\delta z| \leqslant (2\lambda + 1)\gamma + 4\gamma_1.$$

4. Working Precisions

The final task of this section is to study the implications of the above analyses for actual working precisions.

In a 32-bit sli representation, two bits are needed for signs and three for the level, leaving 27 bits for the index. From AO we see that, in order to preserve the expected precision in large arithmetic,

either

$$\gamma = 2^{-30}, \qquad \gamma_1 = 2^{-37}$$

or

$$\gamma = 2^{-31}, \qquad \gamma_1 = 2^{-35}$$

would suffice. With $\gamma = 2^{-31}$, the corresponding values of γ_1 for 'mixed' and 'small' arithmetic are 2^{-34} and 2^{-36} respectively.

There is no obvious necessity for the same working precisions to be used for all operations in a hardware implementation, since these could be executed on different parts of the chip. However, in a software implementation, it may be desirable that, whenever (say) a sequence $\{c_j\}$ is required, the precision should be the same. The values $\gamma = 2^{-31}$ and $\gamma_1 = 2^{-36}$ would suffice in all cases.

It is also necessary, of course, to consider the number of bits which may be needed *before* the binary point: this question is resolved by the bounds given in note (v) of Section 2. One such bit will always suffice for each of a_j, b_j, α_j, and β_j, while two will be enough for c_j except in some cases of small subtraction. The h_j may also need more. The bounds given for c_1, h_l, and h_1 show that, in *all* cases, it will be sufficient to allow c_j and h_j six bits before the point when $\gamma > 2^{-90}$, while five will suffice if $\gamma > 2^{-44}$. (In fact only the first computed element of the $\{c_j\}$ and $\{h_j\}$ sequences can ever need this number of leading bits; the second needs three at most, and the third, two.)

Acknowledgement

The authors are pleased to acknowledge several helpful discussions with and comments from F. W. J. Olver.

References

Clenshaw, C. W., & Olver, F. W. J. 1984 Beyond floating point. *J. ACM* **31,** 319–328.

Clenshaw, C. W., & Olver, F. W. J. 1987 Level-index arithmetic operations. *SIAM J. on num. Anal.* **24,** 470–485.

Clenshaw, C. W., & Turner, P. R. 1988 Root-squaring using level-index arithmetic. (In preparation).

Lozier, D. W., & Olver, F. W. J. 1988 Closure and precision in computer arithmetic. (In preparation).

Turner, P. R. 1986 Towards a fast implementation of level-index arithmetic. *Bull. IMA* **22,** 188–191.