BIT 5 (1965), 37–50

QUASI DOUBLE-PRECISION IN FLOATING POINT ADDITION

OLE MØLLER

Abstract.

The loss of accuracy incurred in adding a small, accurate quantity to a larger one, using floating point addition, can be avoided by keeping account of a small correction to the sum. This is particularly valuable in machines which perform truncation, but no proper round-off, following arithmetic operations. In the first part of the article the details of the method are discussed. In the second part the effectiveness of the method is shown in an application to the step-by-step integration of ordinary differential equations.

1. The method.

We shall investigate the addition $u+v$ of two floating point numbers.

To this end we introduce the notion of level intended as a shorthand for characterizing the relation of the exponent of numbers, *i.e.* we say lower, equal or higher level according as the exponent is less, equal or greater. Thus in case of two numbers, a and b, we write in an obvious manner:

$$\mathrm{lev}(a) < \mathrm{lev}(b), \text{ or } \mathrm{lev}(a) = \mathrm{lev}(b), \text{ or } \mathrm{lev}(a) > \mathrm{lev}(b)\,.$$

We further think of a floating point arithmetic using:

1) true complement representation of negative numbers
2) the interval of the mantissa: $-2 \leqq \text{mantissa} < -1$
   $1 \leqq \quad - \quad < 2$
3) simple truncation of digits from the right in the lower level addend, *i.e.* no correct round-off of this is performed before addition.

When writing down numbers we shall often illustrate the digits in using a mantissa of 5 significant figures:

a: $a_s a_0 . a_1 a_2 a_3 a_4$, a_s being the sign-bit.

The precise aim of the present investigation is to give ALGOL statements that pick up the total loss of bits from the addition statement $s := u+v$ in the form of a definite correction, c.

Ex. u: 01.0110×2^{0}
v: 01.0011×2^{-2} $\Big\}$ s: 01.1010×2^{0} *i.e.* the loss is given by the correction 01.1000×2^{-5}.

We shall in the following suppress the strict floating point notation and freely make use of fixed point notation. Thus the above example takes the appearance:

u: 01.0110
v: 00.010011 $\Big\}$ s: 01.1010
correction: 00.000011.

First we present the idea that underlies the whole of the following. Consider the effect of the following ALGOL statements,

$$\left.\begin{aligned} s &:= u+v \\ v1 &:= s-u \\ u1 &:= s-v1 \end{aligned}\right\}. \tag{1}$$

A little consideration of the results of this shows that though $u+v=s$ may be false it is to be expected that $u1+v1=s$ will always be true. We shall not give the proof of this fact but point at the leading idea namely that $v1$ and $u1$ have been truncated precisely to have only zeros to the right of what are the significant positions of s. On the other hand note that $v1+u=s$ is not always true owing to the possibility that $\mathrm{lev}(s)$ is greater than $\mathrm{lev}(u)$. This is the reason why $u1$ is introduced.

This suggests that the correction to the sum s might be evaluated as follows:

$$\left.\begin{aligned} ev &:= v-v1 \\ eu &:= u-u1 \\ c &:= ev+eu \end{aligned}\right\}. \tag{2}$$

We shall now go through a thorough testing of this idea, which despite its simplicity has pitfalls in it.

We have to distinguish between $|v|<|u|$ and $|v|>|u|$ and shall fix things in assuming $|v|\leqq|u|$. Although the procedure will work even if $|v|>|u|$ it is $|v|\leqq|u|$ that gives the most clear and useful process, because for $|v|<|u|$ the principal term is ev and (as could be expected) eu is zero in most cases, while if $|u|<|v|$ we evaluate ev and eu, i.e. $v1$ and $u1$, in the wrong order, so to speak.

Formulae (1) *and* (2) *in combination with the condition* $|v|\leqq|u|$ *define a process which shall be named* **Process A.**

The discussion is split up according the level of s.

Before we go into details we shall introduce the notations:

1) By subscript s we indicate sign-bits.

2) The digits of

u: $u_s u_0 . u_1 u_2 u_3 u_4$

3) The digits of

v: $v_s v_0 . v_1 v_2 v_3 v_4 x_1 x_2 \ldots$

4) The number built up by the digits of v that belong to the level of u

$v0$: $v_s v_0 . v_1 v_2 v_3 v_4$

5) The cut-off part of v

Δv: 0 0 . 0 0 0 0 $x_1 x_2 \ldots$ always being non-negative

6) The sum s taking one of the following three appearances

$$s: \begin{cases} s_s s_0 . s_1 s_2 s_3 s_4 & \text{for } \operatorname{lev}(s) = \operatorname{lev}(u) \\ s_s s_0 s_1 . s_2 s_3 s_4 & \text{for } \operatorname{lev}(s) > \operatorname{lev}(u) \\ s_s s_s . s_s s_s s_0 s_1 000 & \text{(for instance) for} \\ & \operatorname{lev}(s) < \operatorname{lev}(u) \end{cases}$$

7) and by E_u we indicate the value of a bit in the last position of u.

We shall also talk of the positions of a double-precision u and hereby is meant the positions marked x

$$\begin{array}{l} u_s u_0 . u_1 u_2 u_3 u_4 \\ x\ x\ .x\ x\ x\ x\ x\ x\ x\ x\,, \end{array}$$

i.e. a number placed in the level of u, but having twice the number of digits following the point (though *not* twice the number of significant figures).

$lev(s) \leqq lev(u)$.

Here $v1 = v0$ because, whether $\operatorname{lev}(s) = \operatorname{lev}(u)$ or $\operatorname{lev}(s) < \operatorname{lev}(u)$, both $s := u + v$ and $v1 := s - u$ take place within the same level, $\operatorname{lev}(u)$; consequently $v1 := s - u$ must be the exact reversal of $s := u + v$ which arithmetically is equivalent to $u + v0$. Further $u1 = u$ and hence $eu = 0$.

However, we cannot be sure that the ALGOL statement $ev := v - v1$ shall yield $ev = (v0 + \Delta v) - v0 = \Delta v$ except if $\operatorname{lev}(v1) = \operatorname{lev}(v)$. Thus we

have to consider the possibility $\mathrm{lev}(v) \neq \mathrm{lev}(v1)$ or, because of $v1 = v0$, $\mathrm{lev}(v) \neq \mathrm{lev}(v0)$.

$\mathrm{lev}(v) < \mathrm{lev}(v0)$ requires that v is negative and of the type

$$\begin{array}{lll} & v: & 11.111111x_2x_3x_4x_5x_6\ldots \\ \text{i.e.} & v0: & 11.1111 \\ & v1: & 11.1111 \end{array}$$

giving ev: $00.000011x_2x_3x_4x_5$ which is not the correct value because $x_2x_3x_4x_5$ at least is short of the last bit of v.

That is: if $-\frac{1}{2}E_u < v < 0$ then $c := ev + eu$ will be in error by those figures in v which are to the right of the positions in a double-precision u plus one position. This would be the state of affairs if truncation before addition was strictly followed. However, if the reasonable idea of simply putting $v = 0$ when $\mathrm{abs}(v) \leqq \frac{1}{2}E_u$ is the built-in process of the machine we get $v0 = 0$, $ev = v$ and hence $c := ev + eu$ is correct (in fact $\mathrm{lev}(v) < \mathrm{lev}(v0)$ has not occurred). As this is the way the computer GIER works and because we actually are aiming at a description valid for GIER, the above case shall be omitted when we later summarize the shortcomings of Process A.

$\mathrm{lev}(v) > \mathrm{lev}(v0)$ requires v positive and of the type

$$v: 00.0000x_1x_2x_3x_4x_5x_6\ldots$$

i.e. $v0 := 0$ and hence $ev = v$
that is $c := ev + eu$ is correct.

lev(s) > lev(u).

Here we shall have to treat separately the four possible combinations of bits in the last figure of $v0$ and u.

I $\left.\begin{array}{l} u: \ldots 0 \\ v0: \ldots 0 \end{array}\right\}$ gives $v1 = v0$
$u1 = u, \quad eu = 0$

If $\mathrm{lev}(v) = \mathrm{lev}(v0)$ we are sure that
$ev = v - v1 = v - (v - \Delta v)$ is reproduced by the ALGOL statement
$ev := v - v1$;
However, owing to v: $xx.xxx0xxx\ldots$
$v0$: $xx.xxx0$
The only possibility that $\mathrm{lev}(v1) = \mathrm{lev}(v0) \neq \mathrm{lev}(v)$ is $v0 = 0$

or v: $\quad 00.0000xxx\ldots$

which never could give $\mathrm{lev}(u+v) > \mathrm{lev}(u)$.
Hence $c := ev + eu$ is correct.

II $\left.\begin{matrix} u\colon \ldots 0 \\ v0\colon \ldots 1 \end{matrix}\right\}$ gives $v1 = v0 - E_u$
$u1 = u, \qquad eu = 0$

If $\mathrm{lev}(v) = \mathrm{lev}(v0 - E_u)$ we are sure that our ALGOL statement will reproduce

$$ev = (v0 + \Delta v) - (v0 - E_u) = E_u + \Delta v$$

which is the correct value.
A little reflection shows, that here (owing to the specific last bits) $\mathrm{lev}(v) = \mathrm{lev}(v1)$ except when $-E_u \leqq v < 0$ or $0 < v < 2E_u$.
Taking into account that $\mathrm{lev}(s) > \mathrm{lev}(u)$ the possibilities that $\mathrm{lev}(v) \neq \mathrm{lev}(v1)$ are confined to $-E_u \leqq v < 0$ or in fact

$$\begin{aligned} v\colon &\quad 11.1111x_1x_2x_3x_4x_5\ldots \\ v0\colon &\quad 11.1111 \\ v1\colon &\quad 11.1110 \end{aligned}$$

giving $ev := 00.0001x_1x_2x_3x_4$ while the correct value $E_u + \Delta v$ obviously must comprise at least the next bit x_5.
Notice that u in these cases has to be 10.0000.
Thus: $c := ev + eu$ will be correct except for the special cases where

$$u\colon\ 10.0000 \text{ and } -E_u < v < 0$$

in which we loose the figures to the right of a double precision u. ($v = -E_u$ could be excluded since x_5 here equals 0).
Again, if v is put to zero when $\mathrm{abs}(v) \leqq \frac{1}{2}E_u$ the interval in question is $-E_u < v < -\frac{1}{2}E_u$, and what we loose is precisely the last figure of v.

III $\left.\begin{matrix} u\colon \ldots 1 \\ v0\colon \ldots 0 \end{matrix}\right\}$ gives $v1 = v0$
$u1 = u - E_u$

In this case $\mathrm{lev}(u1) = \mathrm{lev}(u)$ and $eu = 00.0001 = E_u$. Further the only possibility that $\mathrm{lev}(v1) = \mathrm{lev}(v0) \neq \mathrm{lev}(v)$ is $v0 = 0$ which not could occur owing to $\mathrm{lev}(s) > \mathrm{lev}(u)$. Consequently $ev = v - (v - \Delta v) = \Delta v$ is reproduced by $ev := v - v1$ and $c := ev + eu$ gives correctly $E_u + \Delta v$.

IV $\left.\begin{matrix} u\colon \ldots 1 \\ v0\colon \ldots 1 \end{matrix}\right\}$ gives $v1 = v0 + E_u$
$u1 = u - E_u$

If $\mathrm{lev}(u1) = \mathrm{lev}(u)$ and $\mathrm{lev}(v1) = \mathrm{lev}(v)$ we get

$$eu = E_u$$
$$ev = -E_u + \Delta v$$

i.e. $c := eu + ev$ takes the value Δv which is correct.
Now we discuss the questionable cases one by one.

$\mathrm{lev}(u1) \neq \mathrm{lev}(u)$ shall not occur because $u1 = u - E_u$ here means the mere removal of an existing last bit and this never alters the level.

$\mathrm{lev}(v1) \neq \mathrm{lev}(v)$. Here a positive v may give rise to the possibility $\mathrm{lev}(v1) > \mathrm{lev}(v)$, and a negative v to $\mathrm{lev}(v1) < \mathrm{lev}(v)$.
v positive.
If $\mathrm{lev}(v1) > \mathrm{lev}(v)$ v must be of the type

v: 00.0111x_1x_2* hence
$v0$: 00.0111
$v1$: 00.1000
ev: 11.1111x_1 i.e. $\sim E_u + \Delta v$ short of the last bit of v.

As to u it has to be of the type

u: 01.1xx1* giving
$u1$: 01.1xx0
eu: 00.0001$\sim E_u$

Thus $c := ev + eu$ will not give the correct value Δv, the error being a truncation of the last bit of v.
v negative.
If $\mathrm{lev}(v1) < \mathrm{lev}(v)$ v must be of the type

v: 11.1011x_1x_2 hence
$v0$: 11.1011
$v1$: 11.1100
ev: 11.1111$x_1x_2 \sim -E_u + \Delta v$ where Δv always will be correct since what we loose is the last bit of $v1$, i.e. zero.

In summarizing the whole of Process A:

$$\left.\begin{array}{ll} s := u + v; & \\ v1 := s - u; & ev := v - v1; \\ u1 := s - v1; & eu := u - u1; \\ c := eu + ev; & \end{array}\right\} \quad \text{(3, Process A)}$$

---

* Note the particular way in which the strings consisting of digit 1 should match each other:

u: 01.11xxx1
v: 00.001111xxx.

and making allowance for the specific treatment of $\mathrm{abs}(v) \leqq \frac{1}{2}E_u$ as e.g. takes place in the GIER computer, we conclude that provided $|v| \leqq |u|$ c gives the exact correction sought for except in the following special cases:

a1 u: $10.0000 \sim -2$ and $-E_u < v < -\frac{1}{2}E_u$ (consequently $\mathrm{lev}(s) > \mathrm{lev}(u)$) causing the error to be the last figure of v.

a2 u: $01.11..1xx..x1$
v: $00.00..011..11x_1x_2x_3\ldots x_f$ (and consequently $\mathrm{lev}(s) > \mathrm{lev}(u)$) in which cases the last figure of v, x_f (sub$f$ for finis), is lost.

Even more briefly we can state that in the rare cases in which Process A is incorrect the error is a cut away of the last figure of v and will never exceed $\frac{1}{2}E_u$. And when the error equals $\frac{1}{2}E_u$ the result is equivalent to using no Process A ($c=0$), i.e. we never do things worse than the bare addition $s := u+v$.

It should be noticed that only when $\mathrm{lev}(s) > \mathrm{lev}(u)$ and combination III or IV occurs need we evaluate eu; in all other cases $c := ev$ is sufficient.

The process could be put in a compact form by writing

$$s := u+v;$$
$$c := \big(v-(s-u)\big)+\big(u-\big(s-(s-u)\big)\big);$$

For the sake of completeness we shall present an extension of (3) which masters cases a1, a2.

What we do is to evaluate $vlabi := v-(v1+ev)$ ($vlabi$ for last bit of v) and add this quantity to c:

$$c := ev+eu+vlabi;$$

Written in the compact fashion c assumes the monstrous appearance

$$c := \big(v-(s-u)\big)+\big(u-\big(s-(s-u)\big)\big)+\Big(v-\big((s-u)+\big(v-(s-u)\big)\big)\Big);$$

In case a2 this is sufficient in the sense that the pair of numbers (s,c) here comprises all information about the addition $u+v$. In case a1 it is the triple $(s,c,vlabi)$ that comprises all information concerning the result from the addition. We confine ourselves to showing by means of examples how this works.

Ex. 1

u:	10.0000		
v:	$11.11110x_2x_3x_4x_5$		
s:	101.111		
$v1$:	$11.11100\ 0\ 0\ 0\ 0$	ev:	$00.00010x_2x_3x_4$
$u1$:	10.0000	eu:	0
$v1+ev$:	$11.11110x_2x_3x_4 0$	$vlabi$:	$00.00000\ 0\ 0\ 0x_5 0000$

s: 101.111, c: 00.00010$x_2x_3x_4$, *vlabi*: 00.00000000$x_5$0000
Note: $c := c+$*vlabi* will give nothing but c.

Ex. 2

u:	01.1101		
v:	00.0011011		
s:	10.000		
$v1$:	00.010000	*ev*:	11.111101000
$u1$:	01.1100	*eu*:	00.00010000
$v1+ev$:	00.0011010	*vlabi*:	00.00000010000

s: 010.000, $c := ev+eu+$*vlabi*: 00.0000011000

Finally we want to give the results when we apply our Process A to cases where $|v| > |u|$. Of course this is not necessary, the problem has been completely solved in the preceding. However, it shall prove useful to try to drop the condition $|v| \leqq |u|$ which we otherwise have to administer for instance in the following way:

if $abs(u) \geqq abs(v)$ **then** $c := \ldots$
else $c := \ldots$

We prefer to stick to the condition $|v| \leqq |u|$ and instead reverse the order in which $v1$ and $u1$ are evaluated, i.e. we consider what we shall call

Process B:

$$\left.\begin{array}{l} s := u+v; \\ u1 := s-v; \quad eu := u-u1; \\ v1 := s-u1; \; ev := v-v1; \quad c := ev+eu; \end{array}\right\} \text{(4, Process B)}$$

We shall not go through all the laborious considerations which, though different from those of Process A, present no new aspects. We just give the results straight away and, as was the case when Process A was summarized, they are valid for machines which in case $|v| \leqq \frac{1}{2}E_u$ replaces v by zero.

Provided $|v| \leqq |u|$ Process B will give the value of c except when:

$b1$ $\mathrm{lev}(s) < \mathrm{lev}(u)$, and (extending the mantissa somewhat)
u negative: $10.111z_1z_2\ldots z_f$
v positive: $00.0001\,1\ldots\,1x_1x_2\ldots x_f$
which causes the last figure, x_f, to be lost provided:
1) not all among the figures $z_1, z_2, \ldots z_f$ are zero
2) not all among those of $x_1, x_2, \ldots x_f$ that are in the level of s are zero.

$b2$ $\mathrm{lev}(s) < \mathrm{lev}(u)$,
u: 01.0000
v: $-1 < v < 0$, 11. . . . . $x_1 x_2 .. x_f$
which causes an error of $+\frac{1}{2}E_u$ in c provided those figures among $x_1, x_2, .. x_f$ that are in the level of s are not all zero.

$b3$ u: 10.0000 and $-E_u < v < -\frac{1}{2}E_u$
which causes the two last figures of v to be lost.

$b4$ u: 01.1111
v: $E_u \leqq v \leqq u$ and of the type $0x.xxx1x..$
which causes c to be in error by $-E_u$.

In summary we have that in the cases b_1, b_3 the error is a defect of the last or the two last figures of v and will never exceed $\frac{1}{2}E_u$. Here we shall not do thing worse than the bare addition itself. In the cases b_2, b_4 on the contrary the correct value of c is contaminated by $+\frac{1}{2}E_u$ and $-E_u$ respectively, and it would actually be better not to make use of c. However, b_2 and b_4 are indeed rare cases.

If we compare the shortcomings of Process B to those of Process A we see the price for having reversed the proper order into which $v1$ and $u1$ should be evaluated: it is the possibility—inherent in Process B—of loosing the last figure of $u1$ or u that is responsible for the cases b_2 and b_4.

2. Application to step-by-step methods for differential equations.

The above systematic treatment has been worked out mainly for the purpose of clearing up all obscure points of a method which has been used at Regnecentralen during the last year and has proved very useful, as shall be demonstrated below.

It is well-known that when a quantity is built up from repeated addition the crude truncating arithmetic has a one-sided, i.e. systematic, effect on the result. This ill effect can be decisively reduced by using one of the Processes A or B above. The way of doing this in case of a step-by-step integration is shown in the following sketch:

$u :=$ initial u;
$c := 0$;
L: $v :=$ (⟨evaluation of v⟩) $+ c$;
$s := u + v$;
⋮

$$
\begin{aligned}
& c := (v-(s-u))+\big(u-\big(s-(s-u)\big)\big);\\
& \vdots\\
& u := s;\\
& \vdots\\
& \textbf{go to } L;\\
& \vdots
\end{aligned}
$$

or put in a compact form:

$$
\begin{aligned}
& u := \text{initial } u;\\
& c := 0;\\
L:\ & v := (\langle\text{evaluation of } v\rangle)+c;\\
& c := \big(v-\big((u+v)-u\big)\big)+\big(u-\big((u+v)-\big((u+v)-u\big)\big)\big);\\
& \vdots\\
& u := u+v;\\
& \vdots\\
& \textbf{go to } L;\\
& \vdots
\end{aligned}
$$

We shall at once comment on the choice of correction process, i.e. $c := \big(v-(s-u)\big)+\big(u-\big(s-(s-u)\big)\big) \sim eu+ev$ taken irrespectively of the absolute magnitudes of u and v.

In fact three ways of proceeding could be considered. If we do not think of a very accurate process we could decide upon using only $c := v-(s-u) \sim ev$, realizing that $eu \neq 0$ will not happen very often. In the general case we would here have to insert the condition $|v| \leqq |u|$ for instance like

$$c := \textbf{if } abs(v) \leqq abs(u) \textbf{ then } v-(s-u) \textbf{ else } u-(s-v)\ ;$$

However, in the case of an integration we are sure that except for quite incidental situations the contribution, v, from the step will be definitely smaller in absolute magnitude than the variable u itself. Thus, in this case we in fact get excellent results from an unconditional use of $c := v-(s-u)$. Next we could think of a more exact process and here the question is if we should stick to Process A or mingle up with B as becomes the result when the condition $|v| \leqq |u|$ is dropped. We shall not try a discussion as to what extend B is inferior to A, but state the intuitive impression that they in practice are about equally good. Finally we could think of a very exact process i.e. taking $c := ev+eu+vlabi$ which yields a double-precision result (in the sense previously given).

While the first process not is very much simpler than the second and while the cases in which process A, B do not work correctly are of very little practical concern the writer suggests that the process given in the sketch be used for our present purpose as well as in the general case.

The results now to be presented have been taken from a treatment of the following intial value problem:

Given

$$y'' = \frac{2}{1-x}y' - \frac{1}{(1-x)^4}y$$

and

$$y(x=0) = \sin(-1);$$

Find values of $y(x)$ in some interval $0 \leqq x \leqq x_n$. The exact solution is

$$y = \sin\frac{1}{x-1}\,.$$

The step-by-step integration was performed by means of the following fourth order Runge-Kutta formulae:

$$dy/dx = f(x,y)$$

$$y_{i+1} = y_i + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\,,$$

$$k_1 = h\cdot f(x_i, y_i) \qquad k_2 = h\cdot f\left(x_i + \frac{h}{2},\, y_i + \frac{k_1}{2}\right),$$

$$k_3 = h\cdot f\left(x_i + \frac{h}{2},\, y_i + \frac{k_2}{2}\right), \qquad k_4 = h\cdot f(x_i + h, y_i + k_3)\,,$$

h being the steplength.

The program was written in ALGOL and run on a GIER computer having a floating point representation of 29 significant binary digits in the mantissa, the range of which is $|\text{mantissa}| \leqq 2$ (i.e. 28 digits in the fraction). This corresponds to 8–9 significant decimal figures.

The following three concurrent integrations have been performed:

I_1 straightforward use of the Runge–Kutta formulae

I_2 the same formulae now having c inserted according the sketch.

I_3 instead of c another mechanism has been inserted which allows of a correct rounding arithmetic in the crucial addition statement: $s := u + v$ (we return to this rounding mechanism below).

Finally the exact solution, $\sin(1/(x-1))$, has been evaluated.

The steplength was put to $h = .00005$ and the integration run over

6400 steps giving $0 \leqq x \leqq .320$, an interval in which the higher order terms are completely negligible.

The results for seven values of x are given in the table below. *It must be emphasized* that the adding up of x, $x := x+h$, has been treated identically for all three integrations, $I_1 I_2 I_3$, and the process has been to use c according the sketch.

Step-number n	x	Exact solution $\sin 1/(x-1)$	(I_1 − exact sol.) $\times 10^9$	(I_2 − exact sol.) $\times 10^9$	(I_3 − exact sol.) $\times 10^9$
100	.005	−.844 175 437	−93	0	+4
200	.010	−.846 885 566	−186	+2	+2
400	.020	−.852 321 573	−376	+2	+2
800	.040	−.863 247 488	−756	+2	0
1600	.080	−.885 216 752	−1505	0	−32
3200	.160	−.928 545 844	−3166	0	−32
6400	.320	−.994 983 369	−6737	−2	−69

The conclusion is:

The crude truncating arithmetic produces errors that are noticeable after 100 steps and grow proportionally to the number of steps at a rate of about $2^{-30} = .93 \cdot 10^{-9}$ per step.

In using the correction c the errors are brought down to almost zero all through the extended integration comprising 6400 steps.

Regarding I_3 the values actually apply to cases where $h = 2\uparrow$ (integer). When using the rounding mechanism for $x := x+h$ our specific $h = .00005$ produces errors three times those above. However, the writer believe that on a whole a correctly rounding arithmetic needs no correction.*

The mechanism for establishing correct rounding is as follows:

$$s \text{ rounded} := s + 2 \times c;$$

where c has been evaluated according to Process A:

$$c := \textbf{if } abs(v) \leqq abs(u) \textbf{ then } \big(v-(s-u)\big)+\big(u-\big(s-(s-u)\big)\big)$$
$$\textbf{else } \big(u-(s-v)\big)+\big(v-\big(s-(s-v)\big)\big);$$

If we think of a computer having the usual rounding process: s rounded $:= u+(v+\frac{1}{2}E_u)$** for $|v| < |u|$ built in, it emerges that, pro-

* It should be noticed that the Process A and B when applied to a rounding arithmetic will give the rounding error as well. This was Gill's [1] original aim (to be mentioned below) and is quite obvious from the basic idea, namely that $ev+eu$ gives the discrepancy between $u+v$ and $u1+v1=s$ *whatever* be the source of this discrepancy.

** This to be understood in the proper way, i.e. in case v is negative the positive quantity $\frac{1}{2}E_u$ is to be added the complementarily notated mantissa of v.

vided our actual computer when $|v| \leqq \frac{1}{2}E_u$ replaces v by zero, our process above reproduces the behaviour of the rounding computer except when:

1) $-\frac{1}{2}E_u \leqq v < -\frac{1}{4}E_u$ causing $s+2\times c$ to be in error by $-E_u$, and if $u=s$ has the mantissa: 10.0000 the error even mounts to $-2E_u$,
2) $\mathrm{lev}(s) < \mathrm{lev}(u)$ and at the same time $-\frac{1}{2}E_u > v$ or $0 < v$ and when moreover we in v have the digit 1 situated two positions to the right of the last figure of u, i.e. in our notation when

$$v\colon\ v_s v_0 . v_1 v_2 v_3 v_4 x_1 1 x_3 \ldots .$$

Under these circumstances $s+2\times c$ is $\frac{1}{2}E_u$ too large,

and finally when

3) the mantissa of u is 10.0000 and $-E_u \leqq v \leqq -\frac{1}{2}E_u$ giving $s+2\times c$ the value u: 10.0000 while the rounding computer yields the problematic result of 101.111.

It should be added that the ill effect from 1) might be removed in using

if $v < 0 \wedge s = u$ **then** $c := 0$;

s rounded $:= s+2\times c$;

Finally there remain a few remarks on how the method was developed.

Gill [1] has developed a variant of his own among the many existing fourth order Runge–Kutta methods. The merit of his formulae was that they brought down the storage requirement and in addition the reduction of accumulation from rounding errors was cleverly built in.

Through a detailed study of his paper [1] the idea to use $v1 := s-u$ was found. However, in working with fixed point arithmetic Gill was not troubled by the automatic shifts when levels are altered, and this in turn means that he need not introduce $u1$ and he did not have to consider all the subtleties we have encountered. Concerning Gill's fourth order Runge–Kutta formulae the writer's opinion is that they are too complex to be recommended nowadays because the reduction of storage requirement they bring about is of little practical concern and because once his trick regarding the rounding errors is understood and has been put into ALGOL statements as shown above it is universally applicable. His method is often referred to in textbooks from the period 1956–62, but unfortunately without proper explanations of the way his trick works*.

* E.g. Romanelli [2] gives rather detailed working instructions which pretend to yield the reductions of rounding errors; however, in fact his procedure gives no such reduction.

The writer is only aware of one other paper on the present subject: Reducing truncation error by programming by J. M. Wolfe [3]. However, his way of treating the problem follows a different line.

Acknowledgements.

I should like to thank several staff members of Regnecentralen for valuable help in many respects. Especially Knud Hansen, Christian Gram and Peter Villemoes have patiently gone through almost all of the reasonings, and I am much indebted to Peter Naur for having improved the rigour of the structure and wording of the present article.

REFERENCES

1. S. Gill, *A process for the step-by-step integration of differential equations on an automatic digital computing machine*, Proc. Cambr. Ph. Soc. vol. 47 1951, pp. 96–108.
2. M. J. Romanelli, *Runge-Kutta methods for the solution of ordinary differential equations*, Ralston and Wilf, Mathematical Methods for Digital Computers 1959, chap. 9, p. 115.
3. J. M. Wolfe, *Reducing truncation errors by programming*, ACM vol.7/6 june 1964 p. 355.

REGNECENTRALEN
ÅRHUS, DENMARK