VARYING LENGTH FLOATING POINT ARITHMETIC:
A NECESSARY TOOL FOR THE NUMERICAL ANALYST

BY

MARTTI TIENARI

TECHNICAL REPORT NO. CS 62
APRIL 17, 1967

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

VARYING LENGTH FLOATING POINT ARITHMETIC: A NECESSARY

TOOL FOR THE NUMERICAL ANALYST

by

Martti Tienari

Abstract:

   The traditional floating point arithmetic of scientific computers
is biased towards fast and easy production of numerical results without
enough provision to enable the programmer to control and solve problems
connected with numerical accuracy and cumulative round-off errors. The
author suggests the varying length floating point arithmetic as a general
purpose solution for most of these problems. Some general philosophies
are outlined for applications of this feature in numerical analysis. The
idea is analyzed further discussing hardware and software implementations.

Varying Length Floating Point Arithmetic: A Necessary
Tool For The Numerical Analyst

1. Introduction.

The concept of floating point arithmetic was introduced to the
digital computer technology in the early 1950's and since then it has
proved to be one of the soundest standards within the scientific com-
puting field. The extensive use of algebraic languages such as Fortran
and Algol is to a great deal based on the easy use of the floating point
number representation Its unquestionable success rests on the facts
that the floating point number representation is easy to understand and
that it has proved to be reasonably foolproof in practice.

Modern computing machinery has shown a trend to be more and more
easily tailored to the needs of particular applications. This has been
made possible by advances in computer technology: fast logical circuitry,
modular construction, microprogramming, etc. At the same time the cost
of the central processing unit has dropped to be a small part of the
overall system price, even when furnished with combined scientific and
commercial capabilities. With these developments in the background,
one is tempted to ask whether the specifications of the floating point
unit of a modern scientific computer are developed as far as possible.
This question is of special interest to a numerical analyst, who is
often severely restricted by the standard floating point representation
in his work to devise new, reliable computer algorithms.

2. Objectives of numerical analysis.

When considering the practical work of a numerical analyst we notice
that there are quite a few different goals he is aiming at. These goals

are the yardsticks with which the value of his work and contributions are measured. The principal quality measures of an algorithm are its performance in terms of speed and core memory usage. The traditionally very important objective of simplicity of an algorithm, while very important in manual computations, has somewhat lost its signficance. This is true as far as the lack of simplicity does not reflect unfavorably in the program reliability.,

In recent years there has been much research devoted to the problems of numerical accuracy, which is not at all a rare troublespot when manipulating ever larger mathematical models. The demand for accurate algorithms is closely related to the desire to create foolproof, automatic computer programs capable of giving correct answers for widely varying input data. In all algorithms reliability is a very desirable property, especially in the building 'blocks of an application program package. There are many examples where a research and development project has been driven wildly out of schedule because the computer programs have failed to produce meaningful results when receiving new data of unexpected characteristics. The most usual design flaws in application programs, a bug left in the program or the Pack of generality in the programming approach, can often be overcome more easily and with less time delay than problems which are due to the negligence of round-off errors.

We could summarize the practical objective of the present day numerical analyst as being to devise algorithms with an optimal speed, memory usage and program reliability trade-off in a given computing environment. The life of the numerical analyst has become much easier

in the last ten years in mind of these goals The increase of the speed
and memory space available in modern computers has made the best contri-
bution to ease the situation in respect to the speed and space economy
The extensive research of numerical methods in the last decade has
resulted in a well organized body of algorithms for many central prob-
lems in applied mathematics, optimized in speed and memory usage.

The computing economy and the dependence of the real world on
computing have in many applications now reached a level where the
quality standards for program foolproofness have been brought, to the
focus of attention" In many cases higher computing cost might quite
advantageously be paid if one could avoid with this payment the indirect
cost of delays and manpower wastage which are unavoidable when a computer
program unexpectedly breaks down

The most important contributions of modern numerical analysis as a
science have been made in improving the quality of algorithms used.
Much knowledge has been gained from the accuracy characteristics of
central numerical methods through both theoretical and practical work.
The methods used to overcome disturbing round-off effects in numerical
algorithms are to a great deal dependent on the facilities available in
computer systems for this purpose. Our objective in writing this report
is to consider the methods which would be available for numerical analysts
if the computer representation of mathematical real numbers could be
performed in a more flexible way.

3.   General purpose developments for the accuracy problem.

In order to anticipate accuracy problems due to cumulative round-
off errors there has been developed a successful theory of algebraic

-3-

round-off errors [16] . One conceptual breakthrough in this theory is to turn attention from the forward error bounds developed for result accuracy estimation to backward error bounds. The importance of the latter concept is based on the possibility of comparing the effect of round-off errors on a common error scale with measurement or approximation errors. The backward method seems to be suitable especially for the analysis of round-off errors in floating point computations.

The automatic error tracking schemes have concentrated so far on three interrelated developments They are interval arithmetic [7], tracing of error either on a deterministic or statistical basis [10], [14] and unnormalized floating point arithmetic [1] . These schemes are by no means easy to use. Also with the exception of unnormalized arithmetic they would use computing time and memory space wastefully. Unnormalized arithmetic has been the most successful of these schemes; the most significant recognition achieved by it has been its inclusion in the IBM 7090 computer system floating point instruction set [4] . The facilities to make it easy to use for significance tracking within an algebraic compiler scheme are still waiting to be devised.

Once realized, the dilemma of how to overcome the accuracy problem has certainly received some attention. Some scientific computers have been furnished with extra long word lengths. This approach, however, causes wastage in computing speed and memory space in most problems. The most economical and practical solution devised so far has been the inclusion of double-length floating point arithmetic in scientific computers. This step has been strengthened by including an equivalent new variable type in the Fortran IV algebraic programming

-4-

language.   In some character oriented computers, e.g. IBM 1620, there exists the possibility to use operands beyond the conventional double length accuracy.   Unfortunately the speed and memory space limitations as well as the way this facility is supported 'by the Fortran compiler have made the usage of this facility much more uncommon than one would expect from its intrinsic value,

Computer users have implemented for some computers an extension of arithmetic in the form of a subroutine package allowing practically unlimited computing accuracy.   The author is aware of implementations for the IBM 360, IBM 7090, CDC 1604 and Elliott 503 computers [12]. This solution has been dictated by some important applications, in most cases by the solution of ill-conditioned polynomial equations.   In these applications the intermediate stage accuracy needed might be 50-100 decimal digits.

The desire of computer users to get rid of artificial accuracy limits is reflected in the specifications of the new IBM programming language PL/I.   This language allows the programmer to define the operand lengths.   It remains to be seen, however, whether this feature will really be implemented in its full generality and with respective support- ing hardware modifications.

4.   Proposal for the generalization of floating point arithmetic,

The main disadvantages of the usual normalized fixed length arithmetics, which are known to the author, are:

1)   inability to respond to an occasional need for higher accuracy,

2) lack of any provision for tracing round-off error,

3) no possibilities to gain memory space and speed advantages when just a few digits of significance are needed.

Some further disadvantages, which are usually easily solved by the programmer are:

4) the limits set by the fixed length floating point exponent,

5) no possibility to use the floating point overflow and under-flow as an adjustable warning mechanism for the needs of certain applications,

6) the floating point number format does not leave any bits free for flagging certain numbers.

We shall outline a solution for the three major problems given above. The minor problems are recorded here just for completeness; their nature and solution is strictly bound with the economies and conveniences of implementing floating point number formats and circuits in computer systems.

An easy solution, which is not our actual proposal, for all the major problems would be to implement in hardware, instruction sets corresponding to several lengths of floating point number representation, say 16, 24, 32, 48, 64 and 96 bits. The programming language, say PL/I, would contain the possibility of defining the accuracy needed and the language compiler would choose the proper instruction set. If accuracies beyond the hardware floating point formats were required, a subroutine package would take care of that case. Significance trace would be performed either using unnormalized arithmetic or more auto-matically as proposed by Nickel [SO] using with the main floating point number a short floating point number to trace continuously the sig-nificance. What would be the disadvantages of this solution?

-6-

One unpleasant thing from the numerical analyst's point of view would be that the jumps between different accuracy levels might be still too large to enable the analyst the free use of the changing accuracy as will be envisaged later in this article. The lack of elegance and economy in implementing instructions for many different floating point formats would limit in practice the enlargement of the present 32 and 64 bit standard to at most 1 or 2 additional word lengths, say 16 bits and 48 bits. The threshold value for the accuracy, at which the transfer to the software implementation is made, would still be pretty low. The user would thus experience a significant computer slowdown when accuracy is required above this level. Also because of the basic need to create programs which could automatically adjust themselves to a certain required result accuracy, the language compiler should be able to allow changing of the accuracy in a dynamic manner. This would cause dynamic recompilations of program blocks during run time or alternatively routing of all floating point computations through an instruction selection subroutine.

The discussion above has served as an introduction to our actual proposal which is the use of the varying length floating point data form for digital computers. The basic feature we will propose would be to make possible an incremental increase in the accuracy of the floating point fraction over a wide accuracy range. A good step size for a byte-oriented computer would be 1 byte or 8 bits. This would require a floating point instruction set capable of performing arithmetic on numbers with fraction parts of length, say 8,16,24,32,40,48,..., 1024 bits. The actual upper limit of the accuracy would, of course,

depend on the hardware implementation. This facility combined with proper programming language facilities would be quite a tool for both controlling and solving round-off error problems In the following we shall discuss the probable feasibility and impact of this proposal from three different points of view: applications of the proposed device in numerical analysis, implementation in computer hardware, and fitting the device into the algebraic programming language techniques Whether the varying length floating point facility should be augmented by unnormalized operations or other means for significance tracking is not investigated in this article.

5. Applications in numerical analysis

In many well-known numerical methods there exists practical and theoretical evidence for the need of high computing accuracy. The need for high accuracy in the intermediate steps of an algorithm does not necessarily have much to do with the fact that the accuracy of the physical measurements is well exceeded by the computer word length. Some numerical problems happen to be so ill-conditioned with respect to the algorithms used to solve them that the digital random noise due to the cumulative round-off errors destroys the real physical significance of the results.

One obvious type of numerical method which leads to this ill-conditioning is one in which rich information from a large physical data aggregate has been packed into a compact form of a few numbers, and subsequently delicate analy i results are derived exploiting this packed information The real physical dependence of the original data

might be a quite stable one, so that the results, if they can be extracted in spite of the round-off noise, would be very valuable and meaningful indeed. On the other hand we meet also synthetic computing approaches where the results are derived by combining large numbers of data pieces, but where balancing of errors occurs and therefore no trouble with round-off phenomena is met.

We mention as examples some of the best known cases with problematic round-off error history: solution of polynomial equations, inverting large or ill-conditioned matrices and solving the respective simultaneous linear equation systems, solution of some eigenvalue and eigenvector problems, least squares fitting with accurate and intercorrelated models In many of these problems even double precision computing has proved to set limitations. The best known practical application with surprisingly high intermediate accuracy needs is the design of communication filters, which includes the solution of ill-conditioned polynomial equations On the other hand, it is well known that in many simulation and data reduction applications the usual eight decimal digit floating point number length is unnecessarily long and causes wastage of memory space and computing time.

We will try to outline different possible philosophies for the application of the proposed varying length floating point arithmetic. The methods are quite intuitive and heuristic, but we feel that a proper theory of varying precision computations could be developed to give a firmer foundation for the design of these methods. It is worth mentioning that the existing different error analysis methods could be brought into useful practical work through the proposed approaches.

As the first generalized application model we consider the case
where we have reasonably good a priori knowledge of the required com-
puting accuracy, either through former experience or theoretical insight.
The computing accuracies would be determined either in the program
writing stage or dynamically based on the data before beginning a
computation.   As an example we mention a hypothetical simultaneous
linear equation systems solver for general purpose use.   If we wish the
information contained in the 5 most significant digits of the data to be
transmitted to the results without digital round-off noise, the following
accuracy rule would be reasonably sure without being too conservative:
[internal computing accuracy of the solution algorithm] $= L = 5 + 2\log_2 n,$

where n = number of simultaneous equations.   This accuracy formula
is devised using information on the Gaussian elimination method
based on both theoretical and practical evidence; see as a reference
Wilkinson [16], p. 108.   Application model 1 is presented in a general
flowchart in fig. 1.   This method might be called, using the terminology
of the control technology, "feed forward digital noise control."
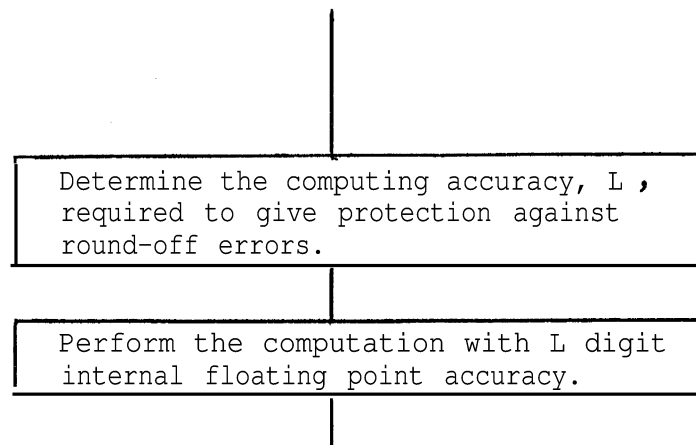
Determine the computing accuracy, L ,
required to give protection against
round-off errors.

Perform the computation with L digit
internal floating point accuracy.

Figure 1: Application model 1 for
varying length floating point arithmetic,
feed forward digital noise control.

-10-

Our second application model is a refinement of the first. It is not generally true that the necessary internal computing accuracy can be determined from the problem data. It is much more common that results and some auxiliary calculations are needed to determine the proper internal accuracy. If the accuracy check after the computation reveals round-off defects in the solution, a new solution process should be initiated with higher computing accuracy. There is always the possibility of a total failure in the first solution which might ruin our decision rule to determine the necessary accuracy increase AL at this stage. Therefore the accuracy check should be made once again after the second calculation. The formula used in model 2 for L and AL would normally be reliable, based on some analysis of the effects of intermediate computing accuracy on the result accuracy. Normally just one solution process would be needed, the numerically ill-conditioned problems going through twice and in exceptional cases more times. We would call-this application model "feedback digital noise control." It is illustrated with a flow-chart in figure 2.
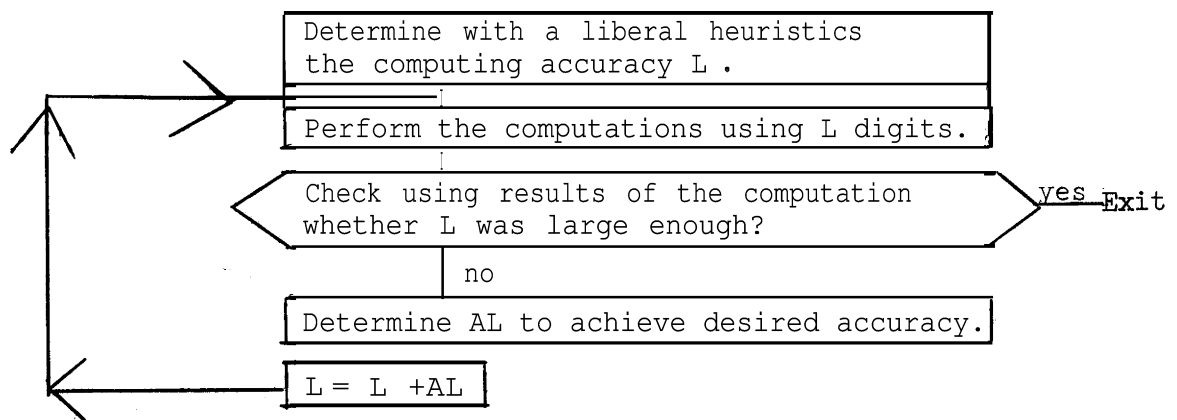

Figure 2: Application model 2, feedback digital noise control.

As an example for application model 2 we consider once again the programming of a general purpose linear equation system solver as discussed in connection with model 1. We might consider that the formula $L = 5 + 2\log_2 n$ would give too high and uneconomical computing accuracies in our particular application field. The formula gives for $n = 100$, $L = 19$. We would like to devise a better, more liberal estimate, which would work well in the majority of cases. A heuristic formula satisfying our intuition might be $L = 5 + \log_2 n$, which gives for $n = 100$, $L = 12$. However, since we no longer have now a firm theoretical background we should make sure that we detect any ill-conditioned equation systems which will not behave regularly. For any computation philosophy desiring reliable results, a check afterwards is useful and in this case it costs just a small fraction of the actual solution time.

The checking method in the linear equation solver might be as follows: given the equation system $Ax = b$ and its solution $x^{(L)}$ using L decimal digit computing accuracy, we compute with $L + 3$ working digits the residual vector $r^{(L)} = b - Ax^{(L)}$. This residual would subsequently be evaluated, component by component, on a suitable reference scale to decide whether the 5 digit significance in the orginal data has been fully exploited. In this case the proper comparison base for the ith component in the residual vector would be

$$d_i = \max \{ |b_i|, |a_{i1}x_1^{(L)}|, \ldots, |a_{in}x_n^{(L)}| \}.$$

The decision for acceptance of the solution $x^{(L)}$ might be made on condition that $|r_i/d_i| < 10^{-5}$ for all $i = 1, 2, \ldots, n$. If $\max |r_i/d_i| > 10^{-5}$, then we should initiate a new calculation using

greater accuracy $L + \Delta L$, where $AL = 1 +$ smallest integer greater than $\log_{10}(10^5 \max |r_i/d_i|)$. There exist methods which perform the new computation on less accuracy that $L + AL$ but we do not consider them here; they do not clarify nor counteract our main theme.

Our third application model is designed for the case when we are unable to devise any reasonable rules for the initial computing accuracy or for the accuracy increase after the first computation. The accuracy behaviour of the problem might be dependent on the actual numerical values in the computation in a way which does not allow us any estimates for the result accuracy or any backward error analysis to judge whether the actual information in the data has been utilized. This philosophy would be also suitable in any computation where available error theory or experience is not relied on or where just for manpower economy, and the need to avoid delays due to the round-off error problems, one is willing to pay for the resulting excessive usage of computer time.

The basic flow of control in this model would be: for an initial accuracy $L_1 = \alpha$ we compute a result which we can think of as a vector $x^{(L_1)}$. Then we increase the accuracy by an increment β to $L_2 = L_1 + \beta$ and compute a new result $x^{(L_2)}$. If the difference of subsequent results $x^{(L_1)}$ and $x^{(L_2)}$, measured with some meaningful method, e.g. using a vector norm, $\|x^{(L_1)} - x^{(L_2)}\|$, is not below our aimed result accuracy level, we continue the computation with $L_3 = L_2 + \beta$. When we finally get $\|x^{(L_i)} - x^{(L_{i+1})}\|$ small enough, we exit from the algorithm with the result $x^{(L_{i+1})}$. We shall call this application model "digital noise filtering loop;" it is illustrated with a flowchart in figure 3.

Some people might object to the decision rule of model 3, which is based on a statistically behaving quantity $x^{(L_i)} - x^{(L_{i+1})}$. In fact in many computations the statistical expectation is theoretically the zero vector for this quantity. We discuss the nature of this decision further in our forthcoming application model 4. In model 3 there might be some advantage in using significance tracking methods for estimating round-off effects, if available. The significance tracking results might help us in the decision to exit the loop; they could even help us to choose the next β more sensibly so we would exit from the noise filtering loop sooner.

Our model 3 is not totally unknown in present day computing practice. It is used in a modified form: L_1 = single length accuracy, L_2 = double length accuracy and the decision rule for exit is replaced by the statement "double length results are as good as we can produce with this computing algorithm."
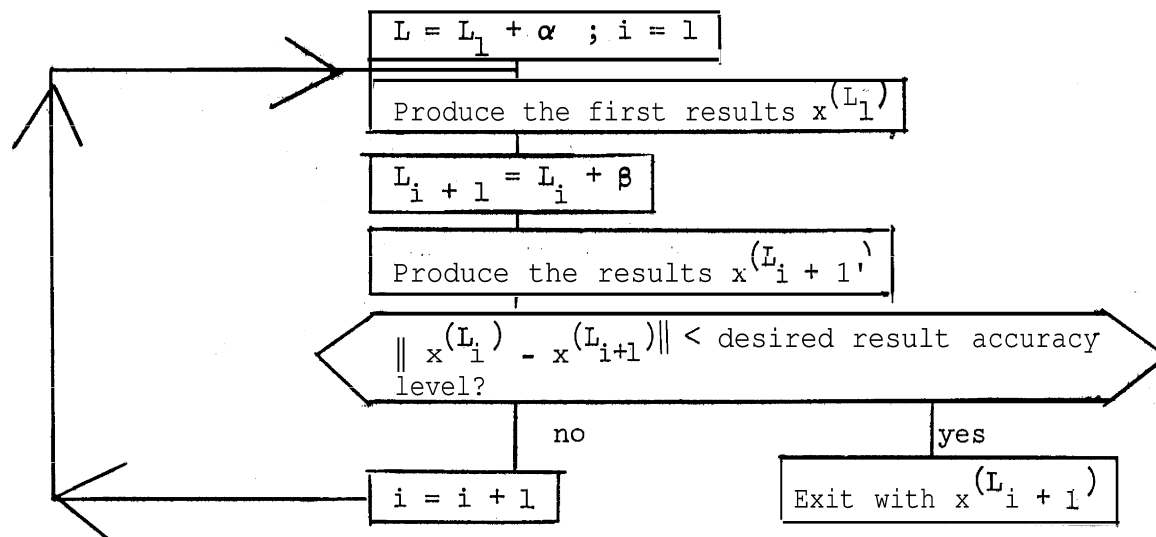

Figure 3: Application model 3 for varying length floating point arithmetic, digital noise filtering loop.

Our fourth application model is used for error estimation. We consider a situation where we have computing results and we are interested to know the effect of round-off phenomenon on the results. We might be unwilling or unable to-'rely on usual mathematically derived error bounds; these might be too conservative for us. To get the round-off error impact on the result accuracy we shall propose one method below which should work in all cases where the order of magnitude of the cumulative round-off effects is directly proportional to the intermediate computing accuracy. Some successful computational experiments designed to prove this assertion have been reported by Ortega [11] for an algorithm for solving the matrix eigenvalue problem.

We should first of all perform the calculation with L decimal digits to obtain our actual results in a satisfactory way. Then we should increase the computing accuracy with say 3 digits and compute the new results. Investigating the coincidence of the results in the leading digits by subtraction would give us approximate error information for our actual results in the form: $\text{error} = K \cdot 10^{-L}$, where $K = K^{(L)} - K^{(L+3)} 10^{-3}$, and $K^{(L)} 10^{-L}$ and $K^{(L+3)} 10^{-L-3}$ are the stochastic errors for the results $x^{(L)}$ and $x^{(L+3)}$ respectively. In most computations $K^{(L)}$ can be considered as a stochastic variable obeying a Gaussian normal distribution with a mean value $m = 0$ and standard deviation σ independent of the computing accuracy used. We assume here that the actual distribution for our specific computation could be constructed allowing L to take on the subsequent values L_0, $L_0 + 1$, $L_0 + 2, \ldots$ with different round-off pattern for every value of L . Can we give any upper bound for σ, when we have just a single observation to rely on?

Using the critical value for χ^2- distribution for one degree of freedom we get with 1% risk of misjudgement:

$$\sigma < \sqrt{K^2/0.000157} < 10^2 K .$$

This limit is achieved by applying the statistical principle that a coincidence which is too good to be true is as abnormal as the incidentally large random deviation. The limit $\sigma < 100$ K is quite encouraging already but there are still additional effects which work in favor of our method. Usually there are several result quantities with errors of roughly the same magnitude and which are not too highly correlated. In these cases if we use a vector norm as the measuring instrument, the error information is dominated by the largest deviation, which gives an estimator for sup σ of a much lower variability. Our assumption of m = 0 is also the worst case for our method. If m $\neq$ 0 which means that there is some bias in the round-off phenomenon, our method would work even better. The round-off bias in $X^{(L)}$ should be of the form $m \cdot 10^{-L}$ so that we would in fact estimate $|m| + \sigma$ which is larger than σ .

The discussion presented above is intended just to support the feasibility of our idea. We do not carry it forward to any completeness here. The conclusion is that multiplying the heuristic error observation quantity derived by our application model 4 by a constant c, which can in many cases be less than 200, we get quite reliable bounds for the actual error.

In most cases it would be advantageous to present as the final results the more accurate ones based on the L + 3 digit computations.

Even the error bound $cK \cdot 10^{-L}$ might be extrapolated for this case as $cK \cdot 10^{-L-3}$. A proper name from the control technology for our method would be "measurement of the digital noise level." Application model 4 is illustrated as a flowchart in figure 4.

The same error estimation procedure presented above might be of use when applied to accurate investigations of the effects of physical measurement errors to the results of an algorithm or a chain of algorithms. We should just choose a computing accuracy which is large enough to remove the digital noise to a level roughly 2-3 decimal places lower than the 'effects of the physical measurement errors. The physical errors to be introduced in the basic data should be simulated with the aid of their assumed external stochastic distributions. The same statistical bounding philosophy which was proposed above would be useful also for this purpose. If we can afford several simulation runs, we might with say 3 calculations with simulated observation errors get a quite realistic and reliable grasp on the real impact of the measurement errors to the results. I would imagine that somebody has done already this kind of investigations though the required error theory and error models are not readily accessible in the literature. This modified use of our application model 4 might be called 'measurement of the physical data noise level in the results.' It is not based on the use of a varying length arithmetic although its systematical and rigorous use would advocate this new concept because of the need to assure that the disturbing digital noise level is a few digits below the physical data noise level.

```
┌────────────────────────────────────────────────────┐
│ Assess an adequate accuracy, L digits, for          │
│ the actual computation.                             │
└────────────────────────────────────────────────────┘
┌────────────────────────────────────────────────────┐
│ Perform the computations with L and L + 3           │
│ digits to produce the results  x^(L) and x^(L + 3). │
└────────────────────────────────────────────────────┘
┌────────────────────────────────────────────────────┐
│ K .10^(-L) = x^(L) - x^(L + 3)                       │
└────────────────────────────────────────────────────┘
```

$$K \cdot 10^{-L} = x^{(L)} - x^{(L + 3)}$$

Do the computation goals allow extrapolation of the error bound to the higher accuracy?

no — Exit with $x^{(L)}$ and error bound $cK10^{-L}$

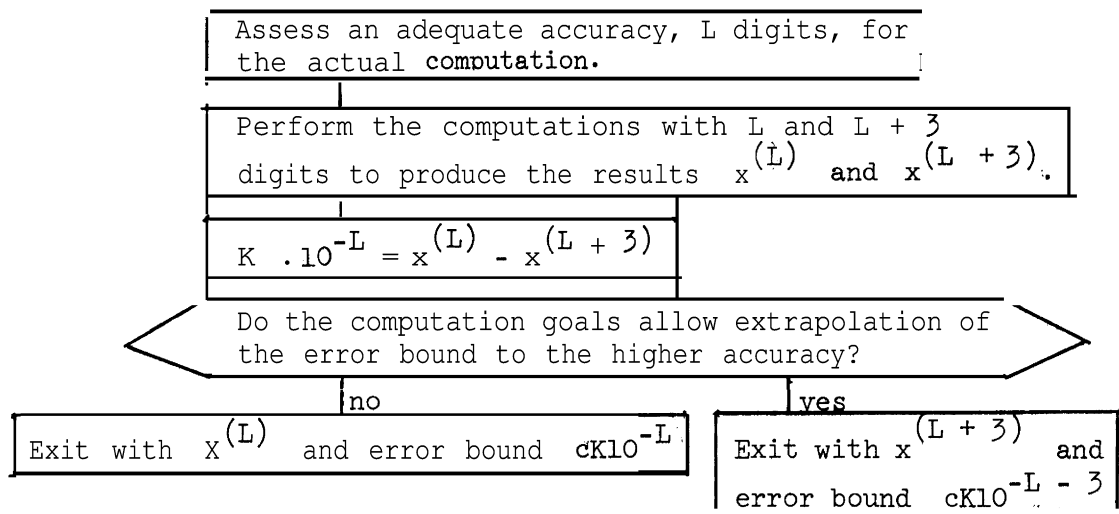yes — Exit with $x^{(L + 3)}$ and error bound $cK10^{-L - 3}$

Figure 4: Application model 4, the measurement of digital noise level.

There exist some new aspects in programming algorithms for the different philosophies 1 - 4 for the use of varying length floating point arithmetic. These aspects are due to the need to devise algorithms which would produce right results over a range of computing accuracies. In all cases we should, of course, establish some absolute upper bound for the computing accuracy which should not be exceeded, the bound depending on our judgement of the possible accuracy needs of a particular computer application, the problem size, and the aimed risk level in the program reliability due to the round-off phenomenon. We shall discuss briefly later in this article, in connection with programming languages, problems due to the new role of the program constants and function sub-routines which must be considered now in a different way than when using fixed length floating point arithmetic. The diagnostic rules for the problem singularities should be reinvestigated, too. The proposed application philosophies would make it necessary to perform these decisions

more carefully than they have been done so far. In many cases where the computing approach constitutes several computations with succes- sively rising internal computing accuracies, the first singularity decisions should be routed to cause a recomputation with a higher accu- racy on the initial data.

6. Some implications to numerical analysis.

The possibility to feel free to use variable length floating point operations would have many fruitful effects on numerical analysis. The basic evaluations for different numerical algorithms - speed, numerical accuracy and memory usage - could be perhaps ultimately reduced to consi- deration of speed and memory usage alone, numerical accuracy being granted. This effect might bring order and simplification to the ever growing mass of numerical algorithms for one and the same problem with different numer- ical accuracy properties. Without any hesitation we claim also that the numerical analyst would feel himself much better equipped for his goal to create more automatic general solution procedures.

The theory of varying precision computing might give rise to a new branch of algebraic round-off error theory. In application model 4 we outlined a new error model for statistical error theory. A philoso- phically satisfying property of this error model is that it allows in principle an unlimited number of independent observations of the error variance estimation rule. Also it should be noted that without a varying length arithmetic it is difficult to apply the results of error theories in practice in an effective way.

Another promising field for varying length floating point computation is the theory of singularity. In computational handling of nearly singular

models, very high computing accuracies are desirable. The numerical decision rules for detecting singularity are today far from complete, perhaps due to the lack of an arithmetic which is flexible enough for rigorous singularity decisions.

Let us consider the economy of the varying precision floating point approach in computer based problem solving. We assume in this discussion that no penalties would be paid because of the generality of the proposed new arithmetic. In other words we assume that the new more flexible arithmetic would perform when applied in the standard, present-day fixed length way as fast as todays standard implementations The operation times are assumed to be nondecreasing functions of the word length L . These functions are probably approximated pretty well by a function of the form $a + bL + cL^2$ with positive constants a, b, c .

In application model 1 we would save computer time assuming that the fixed word length is on an average too long for the problem require- ments. In addition we could provide for the computing client the extra service of assuring a predetermined cumulative round-off noise level. In our model 2 we would also save similarly, at least if we assume that we are able to devise heuristic precision formulas, which on the average are better than $L = 8$ and $AL = 10$. The second computation with a higher accuracy would be also necessary in fewer cases in our scheme than in the present day fixed length approach. If the second computation is needed because of the round-off dangers, an extra benefit is gained in that the computer customer would save one communication cycle with the computing center. This results in significant savings in manpower, computer time and overall problem solution time.

Generally speaking it appears to us that the third and fourth of our application models would result in increased computer time usage. The extra cost incurred should, of course, be motivated by the needs of the application. This extra cost could be thought to be an insurance premium paid for protection against the perils of the round-off phenomenon. In fact the philosophies 3 and 4 are not available today. The applications where these approaches would be adequate are now run with extensive man/machine co-operation.

In some cases the possibility to move flexibly to our strategies 3 and 4, when necessary, would result in large savings in the overall costs of problem solving. The author knows one technical development project depending on computing services where 6 months yere used mainly to fight the accuracy troubles by careful programming using double length arithmetic and when this was not successful another 6 months were required to avoid the trouble using new computing methods. The losses were counted in man years of engineering and programming talent plus a delay of one year in the product development schedule. The computer time budget of the project was a negligible cost compared to these indirect losses. All economical considerations indicated the desirability of application philosophies 3 and 4; unfortunately they were not available because of limitations in computer technology.

7. Implementation in computer hardware.

The author believes that flexible floating point arithmetic should not be too difficult and expensive to implement within a computer with a microprogrammed instruction set. The author of this paper has participated in the construction of a software implementation for the essential

-21-

features of varying length floating point arithmetic [12]. Another

argument supporting the feasibility of varying length floating point

arithmetic is the fact that it has been in effect implemented, although

in a too restricted manner, on the IBM 360/44 computer. This computer

[5] has a rotary switch on the system control panel where the user can

choose the computing accuracy for the double-length floating point

instructions in the range 32, 40, 48 or 56 bits. This feature is

motivated in the machine manual [5] by hinting to the possibility of

gaining speed when the full double-length floating point accuracy is

not needed.

The main system problems in hardware implementation of our proposal

would be the wise selection of floating point instruction and number

formats, the way the problem of operands of different lengths is handled,

the optimal dimensioning of the floating point registers and the internal

decisions affecting the speed performance of the proposed feature.

A possible solution for the instruction format would be to attach

to the floating point unit a new register called the "result accuracy

register." The result accuracy of a forthcoming operation would be

defined by loading the register using a special "accuracy load" instruc-

tion with the desired number of bits or bytes for the floating point

fraction. In the floating point number format it might be desirable

to use a smaller exponent accuracy than 8 bits when the fraction part

is exceptionally short, and more bits for the exponent in the high

precision calculations. We do not know how this can be conveniently

formated, however, without causing restriction for the operations between

operands of different fraction lengths.

We sketch a varying length operand handling scheme. This scheme would lead to a quite satisfactory speed economy and would require no flagging of the operand tails. The arithmetic instruction set would be constructed to work only with numbers loaded in the floating point registers. The instruction set would assume that the result and operand length are the same. It should, however, be possible to gain speed advantages, potentially available, when the operands are shorter than the result accuracy. The microprogram performing floating point arithmetic could, for example, begin the operation by scanning the operands from the least significant end and recognize the zero bit string on the tail of a short operand,

It is evident that the floating point registers for the operands should be quite long to make the varying length floating point operations fast also in higher precisions. Compiler handling of the register over-spill in cheaper storage, without denying this possibility, might be a problem. It would be ideal from the systems programming point of view if the microprogram could control the necessary subroutine branching process in that case. This could be achieved by the following arrangement. When the accuracy register indicated too large a number for the floating point operand register, the load instruction would be interpreted as an operand address load instruction. The subsequent arithmetic operations would initiate an interrupt to the supervisor program which would perform the necessary subroutine entries, The space reservations in the main core memory should be performed by the object time block entry mechanism, if the floating point register overspill was anticipated.

The length of the floating point registers would be a crucial
decision when planning the performance of a variable length floating
point arithmetic. The manufacturer should offer several different
floating point register sizes, e.g. 100 bits, 300 bits, 1000 bits,
with rising cost, preferably with a provision to add extra capacity
later if needed. Efficient coding of the subroutines handling the
operands exceeding the floating point register capacity would be impor-
tant. For this purpose the hardware implementation should include
some specially designed instructions operating on bit strings of length
up to the floating point register capacity. The generality of varying
length arithmetic might be wasteful on short fraction lengths. This
performance defect could be cured by preparing special independent
sections in the floating point microprogram for standard short operand
lengths. The selection of the microprogram sector could be based on
the content of the accuracy register. This superspeed feature might
be subject to a special price in the books of the scientific computer
salesmen!

8.   Varying length operands within the Algol language.

The concept of varying length arithmetic will not be feasible at
all if its use is not made possible within the major programming languages.
There should be no problems which cannot be readily overcome when this
new arithmetic is introduced to the programming languages, Algol and
PL/I. The compilers should be, of course, redesigned but applying
similar techniques as before. We consider the Algol language first
because it is, so far, a better known and more used language than PL/I.

The Algol language must be considered from several aspects to see the impact and the problems caused by the proposed new computer arithmetic. We shall consider first its use as an algorithm publication and program exchange standard. The new language features enabling the programmer to express his decision on the appropriate computing precision will be considered next,. as well as the prerequisites for their economical implementation. Some indirect effects due to the existence of the variable, unlimited length operands are then discussed. These discussions cover program constants, function and input/output procedures.

The operand precision problem in floating point arithmetic has been investigated in order to improve the Algol language [8], which awaits a major revision, being almost unaltered since 1960. According to an idea mentioned in [3] this problem could be solved by introducing new variable types: long real, long long real, etc., into the language. This solution would have from the point of view of our application models 1-4 two basic inflexibilities. Firs%, it would no% allow any dynamical precision changes, which would be essential for the applications. Also it would provide higher accuracies in unneccessarily large increments and would give no provision for speed and memory savings due to the use of very short operands. This straightforward solution would also contribute unfavorably, as noted in [3], to the elegance of the Algol language.

The well known problem due to the dependence of Algol implementations on computer word length plagues to some degree the people making practical use of the published Algol programs. The dependence on the word length comes from program constants or through some implicit dependence. It

is possible to program mathematical algorithms as procedures in a way which minimizes the word-length dependence. This particular programming style has, however, the drawback that it makes the respective procedures more complicated for the user by pushing all the decisions associated with the computing accuracy to the procedure user. This must happen often in a way which no longer makes it possible to consider the procedure as a black box but requires the procedure user to go through the working mechanism of the algorithm. When computer users exchange whole application packages this dependence on the word length is almost impossible to avoid and in practice it is a real trouble indeed. However, one of the basic goals of the Algol language has traditionally been, and should continue to be, the independence of the particular computer implementation as much as is feasible.

It would be an ideal situation if the new revised Algol language could be designed on the assumption that the variable length floating point arithmetic would be available on scientific computer hardware. This starting point could lead to a successful solution of a principal defect in Algol 60, the ignorance of the fundamental role which round-off phenomenon and computing accuracy are playing in every algorithm based on the use of floating point arithmetic.

1% would be possible to repair the accuracy problem simply by declaring in the Algol language all real variables and arrays in a new way: <u>real</u>(n) a, b, c; <u>real</u>(n) <u>array</u> d[1:10]; where n would be an integer constant or expression specifying in decimal digits the minimum significance of the declared operands. It seems to us, however, that if we want the new Algol to be more economical for the user, some information

of the maximum accuracy to be used would be necessary. This would be
furnished to the compiler if the declaration were given in the form:
real(n,p) a, b, c; **real**(n,p) a<u>rray</u> d[1:10]; where n = accuracy of the
operand, p = greatest n to be allowed, n and p are integer constants
or expressions. The arithmetic statements should be evaluated using the
highest accuracy occuring in the operands and the result finally rounded
to the length of the left part variable.

The author cannot accept the criticism presented in [3] of the
decimal representation of accuracy in the language. For algorithm publi-
cation and program exchange purposes a standard accuracy communicating
system would be desirable. The decimal number system is a standard which
is unremovable from our mathematical education. The conversion of the
accuracy specification to different machine representations should no%
be too difficult if we agree on the decimal system. A formula for the
conversion rule for a pure binary machine could be the following: number
of bits in the floating point fraction = 3.32 X decimal accuracy + a
positive implementation convenience allowance. The maximum allowance for
the deviation of an implementation from the decimal equivalent should
be agreed upon. We propose 8 bits as the maximum deviation as this is
compatible with the most popular information organization style of the
contemporary third generation computers.

Let us now consider the operand declarations. We can distinguish
five different modes of accuracy specifications:

   1) standard operand length,

   2) nonstandard fixed length operand,

   3) dynamic   accuracy with fixed upper bound,

4) dynamic accuracy with dynamic upper bound,

5) dynamic accuracy without any upper bound.

I think that case 1, where the programmer would need not specify any accuracy at all, could be omitted in the Algol language. This would be a recommendation consistent to the principle of explicitness as pursued in [3]. Case 2 would be the normal mode of accuracy declaration in Algol. All variables with the same precision should be grouped together in a declaration of the form: real(8) a, b, c;. The programmer could avoid accuracy pitfalls by using longer operands and gain speed and save memory space by using shorter operands. Case 3 would allow dynamic accuracy changing without dynamic memory allocation. Present day compiler technology is probably not able to exploit the slight difference between case 4 and 5. We differentiated between them just to point out that we would propose both forms of declaration real(n) a; as well as real(n,p) a; to be permitted. This recommendation comes from the desire to honor the principle of minimum exceptions Case 4 exists in our proposal because of our desire to include case 3 which allows the compiler to generate efficient code with dynamic accuracy characteristics Case 4 might be also useful when considering procedure publication practices

The program constants present a problem when computing with dynamically changing precision. In some calculation, e.g. the transcendental constant, π, might play such a role that it would be meaningless to perform calculations beyond the accuracy given for the constant. Because the constants perform from the compiler point of view similar functions as the operand identifiers, the constants should be divided

into two classes: 1) constants containing digits up to the equivalent amount of the operand identifier maximum length, 2) constants **exceeding** these limits. One possible and natural solution would be to introduce to the language a constant declaration **statement**, the use of which would be obligatory for constants exceeding the operand name length. The functions of a declaration <u>real</u>(n) <u>constant</u> Pi(3.14159265); would be to assign a storage space equivalent to a 9 decimal digit floating point representation for the value of the real identifier Pi, to define the length of the operand Pi in an equivalent way as <u>real</u>(n,9) Pi;, to give the variable Pi the value 3.141 . . . in n **decimal** digits and to block access **of** the program to the location Pi by forbidding the appearance of Pi in the left side of an assignment statement.

Another new problem would be the implementation of the elementary functions which are usually evaluated with optimized truncated power series. New methods should be devised for these routines **working in** a large range of accuracies. A basic problem would be to devise methods which would be fast enough for short operands and accurate enough and not too slow for long operands. As an example, we sketch a possible method for the exponential function. Write $e^x = e^y \cdot 2^\alpha$ where

$0 \leq y \leq \beta < 1$. Compute $e^y = \sum_{\nu=0}^{\infty} \frac{y^\nu}{\nu!}$; $e^x = (\ . \ . \ (e^y)^2)^2 \ ... \ )^2$. The computation should use intermediate precision of $(n + \gamma)$ digits. We should further reason out an optimal decision rule to choose $\alpha, \beta,$ and y based on some assumption concerning the demand distributions of the argument x and the result precision n .

The input/output procedures in the **Algol** implementations would in principle need no amendments because of the varying length arithmetic. The whole idea of this device focuses on the possibility of controlling the effect of round-off errors. The user's data, as well as his accuracy needs in the results, do no% exceed the accuracy range available today. However, for storing the intermediate results, **to** ease program debugging, and for research in numerical methods it would be convenient to also have variable length input/output routines. This would result in the redesign of the existing routines.

9. <u>Considerations for the programming language PL/I.</u>

After a superficial glance a% [6] it appears that PL/I would allow all the features that we want. There is a provision **to** declare the precision at will if the programmer wants to avoid the standard default accuracy. This standard is implementation dependent, **e.g.** for an IBM 360 PL/I implementation [15], it is equivalent to at leas% 6 decimal digits. The programmer **specifices** the operand accuracy when declaring a real floating point variable including, among the other possible attributes, a precision attribute. For example DECLARE A FLOAT(12) specifies the variable A as a 12 digit floating point variable.

The programmer must, however, notice **that** the compiler of a particular PL/I implementation is free to perform the space reservations and %he actual computations using any suitable floating point format exceeding the programmer's accuracy specification. Neither is the precision attribute included in PL/I in the features which are allowed to be exploited in a dynamic manner in program block entries at object time.

The concept of dynamic data length exists within PL/I; it applies to the string data. Let us consider whether the varying length string data control concepts would be suitable for generalization to the flexible length floating point numbers.

The basic difference between the floating point fraction length control and the PL/I string data length handling philosophy is that the former must be program controlled, whereas the latter is designed to be data controlled. One goal in the design of the VARYING feature for the string data seems to be parallel to the ideas featured in the block entry mode 3 of our Algol operand declaration proposal. Both approaches enable the flexible size fluctuating of varying length data without losing the possibility of static storage allocation. The string data length control philosophy would be suitable for varying length integer and rational arithmetic (infinite precision arithmetic)--which would be useful concepts for discrete numerical analysis--but we cannot conceive any easy method to assign automatically a natural accuracy for varying length floating point results. Therefore the other feature reserved for string data length control, the possibility to se% the maximum length of a string at object time, seems to be the only one which we can make use of. A slight variation to the floating point precision attribute would be desirable, if we want to minimize the speed wastage due to dynamic precision fluctuations.

Considerations presented above lead to the recommendation that the parameter N would be allowed to be defined in the precision attribute (N) a% the object time. In order to achieve object time economies this feature should be supplemented by a possibility to specify the upper

-31-

limit for the accuracy.   We come so to the following language convention "The precision attribute (w,d) of the floating scale is interpreted as follows: w specifies the precision of the floating point number during the object time,   d gives the upper limit of the precision, w and d may be constants or expressions.   If d is no% given, it is assumed that w = d ."

A change to the PL/I language implementation philosophy would be needed, too, if we really want to benefit from the proposed application models 1-4 presented earlier in this article. The implementation should follow the programmer's accuracy specification. To be explicit, some convention like the following would be needed.   "The accuracies used by PL/I implementation in storing and computing floating point numbers should follow each other by increments of not more than an equivalent of 8 bits. For a particular accuracy of a program it should be assigned the nearest larger precision to the equivalent of the accuracy in the pro-grammer's specification.' This convention would still allow a binary implementation to follow a byte structure.   The programmer could also be sure that if he increases the accuracy in his computation by 3 decimal digits the round-off pattern is changed.

Is it feasible to implement our ideas in the present generation of computer hardware without supporting special hardware facilities? In a PL/I implementation for a computer without hardware floating point facilities the ideas would be useful to consider immediately. The speed economies achievable might be worth earnest considerations. In a computer with floating point hardware and byte organized memory, like the IBM 360, the accuracies 8, 16, 24, 40, 48 and 56 for the fractional part

would be available for fast computing. For the accuracies above 56 bits there should be a set of subroutines available in the language for long precision arithmetic and elementary functions.

The extension of PL/I to the generality we are aiming at could be accomplished honoring the upwards compatibility principle. The existing PL/I programs could be run with the same speed efficiency using the extended language compiler. To achieve this aim the new compiler should be able to choose between two modes of code generation. For every block entry in the source program a decision would be made whether or no% any non-standard accuracy features are used.   The arithmetic on the nonstandard or dynamic precision variables would be compiled using a floating point arithmetic selection subroutine. This subroutine would perform arithmetic on the fraction accuracies 8, 16 and 24 bits using single length floating instructions and rounding the result to the right result precision.   The computations with the fraction lengths of 32, 40, 48 and 56 bits would be performed using the double length instructions, and fraction lengths 64, 72, . . . should be handled with the aid of special software subroutines.   The user should be informed of the standard constant accuracies and of the full efficiency of the code %he compiler would generate when he uses one of them.   He cannot benefit from other accuracies at the present time anyway, because in most compilers all fraction accuracies below 24 bits are handled equivalent to 24 bits internally and all accuracies between 25 and 56 bits are equivalent to 56 bits fraction accuracy.

The introduction of varying length arithmetic without hardware support causes an extra burden, especially an extra allowance of core

memory space for the PL/I compiler. The object time economies achievable
today are due to the storage space savings and application flexibility
gains without any speed savings if the hardware floating point unit is
available. It is therefore doubtful whether these recommendations are
acceptable today when the PL/I compiler writing is difficult anyway.
The practical utility of our application philosophies would need more
concrete case examples to act as a driving force towards these goals.
The problem is that motivating application cases will not become
available until somebody constructs a compiler to make the programming
of these applications feasible.

The best way to get these ideas properly investigated would be to
get some university compiler group interested in our application models.
This should happen in a place where numerical analysis research is
pursued, We believe that the application potential available through
this kind of compiler is worth exploring for the benefit of numerical.
analysis.

10.   Conclusions.

We claim that round-off error differs in a fundamental manner from
other uncertainties involved in computing. It can be effectively fought
using computer based **means.** This conclusion is more optimistic than
many earlier assertions [2], [13] concerning the nature of the error
problem. To promote this conviction we propose the return to the use of
the term "digital noise" (or "processing noise" or "computing noise")
as a synonym for the term cumulative round-off error as proposed in [9].
This would distinguish round-off error from approximation errors and

would also underline the responsibility of the computer system designer for this error category.

We hope that the computer manufacturers would consider seriously the inclusion of varying length floating point arithmetic in their scientific hardware and software. This feature when powerfully implemented might prove to be an excellent sales argument for a new computer intended for the scientific computing market. The economies from which the users would benefit with this feature are:

1) Better matching of computing precision to the actual needs resulting in speed improvements and core space savings

2) Running time savings when the fastest available algorithms could be used also for occasionally numerically ill-conditioned problems.

3) Savings in the overall problem solving costs when numerical accuracy problems can be handled with straightforward philosophies.

4) The possibility to use brute force in solving round-off error problems when delays in the computing service appear to cause unreasonable indirect costs.

5) A better overall quality in the scientific computing services from the numerical precision point of view.

It seems not to be generally known that varying precision floating point arithmetic would provide a more elegant and practical scheme to control the round-off errors than the earlier error tracking schemes. It has the overriding practical advantage that it does not only warn the user of the round-off error problem, but it also helps him solve it. In order to exploit this philosophy some new research on the numerical methods would be desirable. This research would be performed with the

aid of a software simulated variable length floating point arithmetic, preferably augmented by an automatic error tracking scheme. With this kind of work in the background it would be much easier to decide whether the proposed new features are worth the extra hardware cost. In any case this research would catalyze new insignts on the effects of round-off errors in computing.

REFERENCES

[1] R. L. Aschenhurst, "Techniques for Automatic Error Monitoring and
Control", Error in Digital Computations, Vol. I, L. B. Rall (Ed.)
New York: John Wiley and Sons (Oct. 1965), p. 43-59.

[2] S. Gorn, "The Automatic Analysis and Control of Computing Errors",
J. Soc. Industr. Appl. Math 2 (Dec. 1954), p. 69-81.

[3] C. A. R. Hoare and N. Wirth, 'Contribution to the Development of
Algol 60", Comm. ACM 9 (June 1966), p. 413-432.

[4] I.B.M. Reference Manual, "7090 Data Processing System", Form
~22-6528-2 (Feb. 1961).

[5] I.B.M. Systems Reference Library, I.B.M. System/360 Model 44,
"Functional Characteristics", Form A22-6875-3 (Jan. 1966).

[6] I.B.M. System Reference Library, I.B.M. System 360 Operating System,
"PL/I Language Specifications", Form C28-6571-3, New York:
International Business Machines Corporation (July 1966).

[7] R. E. Moore, "The Automatic Analysis and Control of Error in
Digital Computation Based On the Use of Interval Numbers",
Error in Digital Computation, Vol. I, L. B. Rall (Ed.) New York:
John Wiley and Sons (Oct. 1965), p.61-130.

[8] P. Naur, (Ed.), "Report on the Algorithmic Language Algol 60",
Comm. ACM 3 (May 1960), p. 299-314.

[9] J. von Neumann and H. H. Golds-tine, "Numerical Inverting of
Matrices of High Order Bull. Amer. Math Soc. 53 (1947),
p. 1021 -1099.

[10] K. Nickel, "Uber Die Notwendigkeit einer Fehlerschranken-Arithmetik
für Rechenautomaten. Num. Math. 9 (1966), p.69-79.

[11] J. M. Ortega, "An Error Analysis of Householder's Method for the Symmetric Eigenvalue Problem", Technical Report No. 18, Appl. Math. and Statistics Laboratory, Stanford University, California (Feb. 1962).

[12] M. Tienari and V. Suokonautio, "A Set of Procedures Making Real Arithmetic of Unlimited Accuracy Possible Within Algol 60", Bit 6 (1966), p. 332-338.

[13] J. Todd, "The Problem of Error in Digital Computation", Error in Digital Computation, Vol. I, L. B. Rall (Ed.) New York: John Wiley and Sons (1965), p. 3-41.

[14] W. C. Wadey, "Floating Point Arithmetic", J. ACM 7 (1960), p. 129-139.

[15] E. A. Weiss, The PL/I Converter, New York: McGraw Hill (1966).

[16] J. H. Wilkinson, Rounding Errors in Algebraic Processes, New Jersey: Prentice-Hall (Jan. 1963).