МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2011
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2011
Proceedings of the Fortieth Jubilee Spring Conference
of the Union of Bulgarian Mathematicians
Borovetz, April 5–9, 2011

LOSS OF ACCURACY IN NUMERICAL COMPUTATIONS*

Mihail M. Konstantinov, Petko H. Petkov

In this tutorial paper we consider possible catastrophic effects of improper use of finite machine arithmetic. Unfortunately, this topic is not well understood even by students in applied and computational mathematics. The situation in engineering and economic specialities is by no means better. To overcome this educational gap we describe the main reasons for loss of accuracy in numerical computer calculations. We hope that the results presented will help both students and lecturers to understand better and avoid the main factors that may destroy the accuracy in computer calculations. The latter is serious – numerical catastrophes sometimes yield real ones, with large damages and human casualties.

Introduction. In this paper we consider some major reasons for loss of accuracy in numerical computations. We restrict ourselves to the real case although most results are directly applicable to computations with complex quantities (MATLAB for example uses complex arithmetics). Often the loss of accuracy may be the result of current or past effects of rounding of numerical data when Finite Machine Arithmetic (FMA) is used. However, even exact computations with uncertain data may be contaminated with large errors. The reasons for such effects must be carefully analyzed and measures to avoid them are to be taken.

The numerical examples given below are prepared using the computer system MATLAB[1], see e.g. [8].

Elements of FMA. A FMA consists of a finite set of machine numbers $\mathbb{M} \subset \mathbb{R}$ together with the rules for performing operations in $\mathbb{M}$, including the rules for transforming (or rounding) real into machine numbers. The set $\mathbb{M}$ contains 0 and is symmetric relative to $\mathbb{R}$. It depends on four parameters: the base b (usually $b = 2$ or $b = 10$), the integer precision $p > 1$ and the largest and smallest allowable exponents $e_{\max}$ and $e_{\min}$. Each machine number $m \neq 0$ has a normal representation $\pm s \times b^e$, where $s = \overline{d_0.d_1 \dots d_{p-1}} = \sum_{k=0}^{p-1} d_k b^{-k}$ is the *significand* (or *mantissa*), $e \in [e_{\min}, e_{\max}]$ is the *exponent* of m and $1 \leq d_0 \leq b-1$, $0 \leq d_j \leq b-1$, $j = 1, \dots, p-1$.

In FMA a number $x \in \mathbb{R}$ is rounded to the nearest machine number $x^* \in \mathbb{M}$ and $x^* = x$ if and only if $x \in \mathbb{M}$. If x is in the middle between two consecutive machine numbers, then it is rounded to the machine number with an even (zero in binary floating-point

*2000 Mathematics Subject Classification: Primary 97N20.
Key words: accuracy, numerical computations, computational catastrophes.
[1] MATLAB© is a trademark of MathWorks, Inc.

FMA) least significant digit d_{p-1} in s, see [9, 1]. Directed roundings toward 0, $+\infty$ and $-\infty$ are also used when interval algorithms are implemented but this subject is beyond the topic of the present tutorial paper.

We shall consider the binary floating point FMA of MATLAB which obeys the IEEE standard [4, 5], see also [2, 3]. It is characterized by three important positive numbers, namely

$$\begin{aligned} \mathbf{r}_{\max} &= 2^{1024} \simeq 1.7977 \times 10^{308}, \\ \mathbf{r}_{\min} &= 2^{-1022} \simeq 2.2251 \times 10^{-308}, \\ \mathbf{u}_{\rm rnd} &= 2^{-53} \simeq 1.1102 \times 10^{-16}. \end{aligned}$$

They may be recovered in MATLAB by the commands `realmax`, `realmin` and `eps/2`. The number $\mathbf{u}_{\rm rnd}$ is said to be the *rounding unit* of the FMA.

A number $x \in \mathbb{R}$ is in the *standard range* of FMA if either $x = 0$, or $|x| \in [\mathbf{r}_{\min}, \mathbf{r}_{\max}]$. In this case x is rounded to the nearest machine number $x^* \in \mathbb{M}$ (with the rule to break ties described above) so that $0^* = 0$ and

$$\frac{|x^* - x|}{|x|} \leq \mathbf{u}_{\rm rnd}, \ x \neq 0. \tag{1}$$

Thus numbers from the standard range are rounded with small relative error of order 10^{-16}, i.e. with 15-16 true decimal digits.

Suppose now that $\diamond$ is an arithmetic operation and that the non-zero quantities x, y and $x \diamond y$ are in the standard range of FMA. Let $(x \diamond y)^* \in \mathbb{M}$ be the result of the machine computation of $x \diamond y$. Then according to the *Main Hypothesis* of FMA with a guard digit we have

$$(x \diamond y)^* = (x \diamond y)(1 + \alpha), \tag{2}$$

where $|\alpha|$ is a small multiple of $\mathbf{u}_{\rm rnd}$.

What happens with numbers $x \in \mathbb{R}$ that are outside the standard range of FMA? There are four possible cases. The story is interesting and not very well known. Denote

$$\delta = \mathbf{r}_{\min}\mathbf{u}_{\rm rnd} = 2^{-1075} \simeq 4.9407 \times 10^{-324}, \ \Delta = \mathbf{r}_{\max}\mathbf{u}_{\rm rnd}/2 = 2^{970} \simeq 9.9792 \times 10^{291}$$

and recall that MATLAB uses the special symbol `Inf` to denote $+\infty$. It corresponds to the IEEE arithmetic representation for positive infinity [4, 5].

- If $|x| \geq \mathbf{r}_{\max} + \Delta$ then $x^* = \pm\,\mathrm{Inf}$ depending on the sign of x and the relative rounding error is formally ∞.
- If $|x| \in (\mathbf{r}_{\max}, \mathbf{r}_{\max} + \Delta)$ then $x* = \pm\,\mathbf{r}_{\max}$ depending on the sign of x and the relative rounding error does not exceed $\mathbf{u}_{\rm rnd}/2$.
- If $|x| \in (\delta, \mathbf{r}_{\min})$ then x is rounded to a non-zero quantity but the rule (1) is no more valid.
- If $|x| \leq \delta$ then $x^* = 0$ and (1) is violated since the relative rounding error is now equal to 1.

Over- and underflows. It follows from the previous section that two major accuracy killers are the *overflow* $|x| \geq \mathbf{r}_{\max} + \Delta$ with $x^* = \pm\,\mathrm{Inf}$ and relative error ∞,

and the *underflow* $0 < |x| \leq \delta$ with $x^* = 0$ and relative error 1. Until recently computer programs simply used to stop when overflow occurred (some of them are still doing so). The famous *division by zero* may also be interpreted as an overflow. However, MATLAB admits division by zero without stopping the computations. For example the commands `1/0` and `-2/0` give `Inf` and `-Inf` respectively.

In MATLAB according to the IEEE standards the symbol `NaN` (from *Not a Number*) denotes mathematically undefined quantities, e.g. `Inf-Inf = NaN`, `Inf/Inf = NaN`, `0/0 = NaN`, `1^Inf = NaN`. Note however that `Inf^0 = 1`.

Loss of exact left–most digits in subtraction. This phenomenon is known as *cancellation* or even *catastrophic cancellation* in view of the destructive effect it may have on the accuracy. This main accuracy killer in computer calculations arises when close positive numbers, say $x > y > 0$, are subtracted and the information coded in their left-most digits is lost. However, the effects and the nature of cancellation are often underestimated and/or misunderstood. The reason for loss of accuracy is *not* the error done in the machine subtraction $(x - y)^*$ itself (when guard digit is used, see [1]). Moreover, when $x < 2y$ (which is usually the case) and $x, y \in \mathbb{M}$, then the machine subtraction is exact, i.e. $(x - y)^* = x - y$.

Suppose that $x = \overline{d_0.d_1 \dots d_{n-1}\xi}$ and $y = \overline{d_0.d_1 \dots d_{n-1}\eta}$ are two machine numbers in a b–base FMA, where the first n digits in the significands are true whilst the digits $\xi > \eta \geq 0$ are uncertain. If we subtract these very exact numbers the result $x - y = (\xi - \eta)b^{-n-1}$ will contain no true significant digit! Note that no errors during the subtraction have been made.

It may be observed that in school and even in some university courses the solution of the quadratic equation $AX^2 + BX + C = 0$, $A \neq 0$, is represented as

$$x_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}.$$

Due to roundings and possible cancellations, this is probably the worst way to solve the equation in FMA.

Another instructive example of cancellation (known in the past as a test to evaluate the rounding unit) is computing the quantity `(1 - 3*(4/3 - 1))/eps`. The exact answer is 0 but the computed result is 1!

Loss of exact right–most digits in summation. In some cases a loss of distant true right-most digits may occur with catastrophic consequences similar to these described in the previous section. This accuracy killer arises when positive numbers x, y of very different size are added. Here all digits to the left of the cancelled ones should be exact. The right–most digits are cancelled due to rounding. If $x > y$ and $y/x \ll 1$ then in the computed sum $(x + y)^*$ (being itself rounded with small relative error of order $\mathbf{u}_{\rm rnd}$) the information about y may be partially or completely lost. In particular, if $y/x \leq \mathbf{u}_{\rm rnd}$ then $(x + y)^* = x^*$ and no trace of y remains in the computed sum. This may be dangerous.

Consider the following (obviously bad but possible) algorithm for computing the Euler constant $\mathbf{e} \simeq 2.7183\dots$, based on the well known expression

$$\mathbf{e} = \lim_{h \to +0} \mathbf{e}_h, \ \mathbf{e}_h := (1 + h)^{1/h}.$$

Omitting the details, there exists an optimal value $h_0 \simeq \mathbf{u}_{\rm rnd}^{1/2}$ for h such that the computed $\mathbf{e}^*_{h_0}$ is true with the least possible relative error of order $\mathbf{u}_{\rm rnd}^{1/2}$. This is not a coincidence: usually the relative step in computational processes such as computing numerically derivatives and solving numerically ODE's, is of order $\mathbf{u}_{\rm rnd}^{1/2}$. Accordingly, the overall relative error is at best of the same order. Attempts to reduce the step may result in a catastrophe.

What happens when h decreases below h_0? The results is a blow up of the relative error $|\mathbf{e}^*_h - \mathbf{e}|/\mathbf{e}$ which, for $h \leq \mathbf{u}_{\rm rnd}$, reaches 0.6321 since for this value of h the computed $\mathbf{e}^*_h$ is equal to 1. The reason is that the quantity $1 + h$ is rounded to 1 and the whole information coded in h disappeared.

Another simple example is the computation of the expression y given by the command `y = (1 + x*eps/2 - 1)/(x*eps/2)` for $x \in (0, 2)$. Instead of the correct answer $y = 1$ the computed result is $y^* = 0$ for $x \in (0, 1]$ and $y^* = 2/x$ for $x \in (1, 2)$. The reason is that $(1 + z)^* = 1$ for $z \in (0, \text{eps}/2]$ and $(1 + z)^* = 1 + \text{eps}$ for $z \in (\text{eps}/2, \text{eps})$.

Also very interesting is the computation of the expressions

$$y_1 = \frac{(1+x)^2 - 1 - 2x}{x^2},\ y_2 = \frac{(1+x)^2 - (1+2x)}{x^2}$$

for small $x > 0$. Of course, $y_1 = y_2 = 1$. But in FMA $y_1 \neq y_2$ and both y_1, y_2 may be quite different from 1. It is useful to make these computations with x close to 10^{-8} with both binary and non-binary values. Now at least some of the students will be slightly surprised and ready to do their own experiments in order to "cheat" the FMA of MATLAB.

High sensitivity of computational problems. Any numerical computational problem (including an infinitely dimensional one) may be formulated as a function evaluation $x = f(a)$, where the data a and the result x are elements of finite-dimensional spaces. Consider for simplicity the case when both a and x are non-zero scalars and the function f is differentiable. Then we have an inevitable error in the function evaluation which may be estimated as follows.

In general $a \notin \mathbb{M}$ and, hence, we may only compute $f(a^*)$, where a^* is the rounded value of a, i.e. $a^* = a(1 + \delta_a)$, where $|\delta_a| \leq \mathbf{u}_{\rm rnd}$. Even if there are no errors in the computation of $f(a^*)$ (a very rare idealistic case!), then within first order terms in $\mathbf{u}_{\rm rnd}$ we have $f(a^*) = f(a) + f'(a)a\delta_a$ and, hence,

$$\frac{|f(a^*) - f(a)|}{|f(a)|} \leq \mathbf{u}_{\rm rnd}\, K,\ K := \frac{|a|\,|f'(a)|}{|f(a)|}.$$

The constant $K = K(f, a)$ is the *relative condition number* of the computational problem $x = f(a)$ and is a measure of its sensitivity. If $\mathbf{u}_{\rm rnd}\, K < 1$ (otherwise there may be no true digits in the computed result $f(a^*)$) then we may expect about $-\lg(\mathbf{u}_{\rm rnd}\, K)$ true decimal digits in the computed solution.

A large value of K is an indicator for possible loss of accuracy. Thus we identify the following three potential accuracy killers when solving computational problems in FMA: *large arguments*, *large derivatives of the evaluated function* and *small function values*. The first one may eventually be neutralized by scaling and shifting of the argument. Examples of evaluation of simple trigonometric functions (such as sine and cosine) with large arguments clearly demonstrate the above conclusions.

An instructive observation here is that the catastrophic cancellation in subtraction of close positive numbers in the implementation of a computational algorithm also leads to small intermediate results.

Large intermediate results. When large intermediate results occur in the implementation of a computational algorithm and the final result is small then large relative errors in the computed solution should be expected. A classical example here is the improper computation of the exponential $x = \mathbf{e}^{-a}$ for $a > 0$ by truncation of the Taylor series for this function in the neighborhood of $a = 0$, namely

$$x = \mathbf{e}^{-a} \simeq \sum_{k=0}^{n} \frac{(-1)^k a^k}{k!}.$$

Computing $\mathbf{e}^{-a}$ in this disastrous way may lead to very large intermediate terms computed with large absolute errors. This may produce large relative error in the computed result for $\exp(-a)$. In this case the large intermediate results are combined with subtractive cancellations (disasters rarely come alone). If we insist to compute the exponential by Taylor series we should at least first find $y = \mathbf{e}^{a}$ and then determine x as $1/y$.

Improper use of residuals in solving equations. Consider the problem of solving the (generally non-linear) equation $f(x) = 0$, where $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}^n$ is a given continuous function. Suppose that there is a solution $x_0 \in \mathbb{R}^n$, i.e. $f(x_0) = 0$. One approach to solve the equation is to minimize the scalar expression (the residual) $F(x) = \|f(x)\|$ since $F(x) \geq 0$ and $F(x) = 0$ if and only if $f(x) = 0$.

Usually to find x_0 exactly is impossible and we look for an approximation ξ such that $\|\xi - x_0\|$ is of order $\mathbf{u}_{\rm rnd}$. This may also mean that $F(\xi)$ is of order $\mathbf{u}_{\rm rnd}$ as well. But if we have two approximations ξ_1 and ξ_2 then which one to choose? Can we use the residuals $F_k = F(\xi_k)$ for this purpose?

Two facts, namely that the function $F : \mathbb{R}^n \to \mathbb{R}_+$ is continuous and that $F(x_0) = 0$ had contributed to the creation of the next harmful and (unfortunately!) wide spread misconception.

Myth about residuals. If $F_2 < F_1$ then approximation ξ_2 should be better than ξ_1 in the sense that $\|\xi - \xi_2\| < \|\xi - \xi_1\|$.

This is a wrong receipt with potentially disastrous consequences. The only case when this assertion is always true is for scalar affine functions f, i.e. for trivial equations. The myth obviously fails for non-linear scalar and vector equations, see [7]. But it also fails for linear vector equations $Ax = b$ with $A \in \mathbb{R}^{n\times n}$ invertible for all $n > 1$. It is easy to find an example in which $A(t) \in \mathbb{R}^{2\times 2}$ depends on a small parameter $t > 0$, the solution $x_0 = A^{-1}(t)b$ does not depend on t and there are two vectors $\xi_1(t), \xi_2 \in \mathbb{R}^2$ such that

$$\|\xi_1(t) - x_0\| = t\|\xi_2 - x_0\|, \ \|A(t)\xi_2 - b\| = t\|A(t)\xi_1(t) - b\|.$$

Thus for $t \to 0$ the accuracy test based on residua is completely misleading. As may be expected, here the condition number $\|A(t)\|\,\|A^{-1}(t)\|$ of $A(t)$ tends to ∞ as $t \to 0$.

So there is a paradox with this myth about residuals. It may be working properly for well conditioned linear equations when the solution is computed with high accuracy and no check is needed. And it may be severely misleading when the equation is ill conditioned and an accuracy check is necessary.

Inappropriate decomposition of computational problems. Let a computational problem $x = f(a)$ be decomposed as $f = f_1 \circ f_2 \circ \cdots \circ f_n$, where some of the functions f_k is very sensitive while the original one is not. Then this is an inappropriate decomposition which may destroy the accuracy of the computed result. The mastership of the numerical analyst is to detect and avoid such decompositions.

Consider for example the problem of finding some (or all) of the eigenvalues of the square matrix A. For many years in the past the method was first to compute the coefficients of the characteristic polynomial $p_A(\lambda) = \det(\lambda I - A)$ of A and then apply some of the (sometimes very sophisticated) numerical algorithms for solving algebraic equations. The above two subproblems (to determine the coefficients of $p_A(\lambda)$ and to solve the equation $p_A(\lambda) = 0$) may be very sensitive even if the eigenvalues of A are not so sensitive relative to perturbations in the elements of A. This may lead to unnecessary large errors in the computed eigenvalues.

The modern approach to find the eigenvalues is to apply the QR algorithm for reduction of A into its upper triangular Schur form S by unitary similarity transformations [2]. Then the eigenvalues of A are the diagonal elements of S. But this is not the end of the story. If for some reason we have to solve numerically an algebraic equation $p(\lambda) = 0$ we no more use the sophisticated algorithms from the past. Rather, we construct the accompanying matrix H of p such that $p_H(\lambda) = p(\lambda)$ and apply the QR algorithm to find the eigenvalues of H. Thus the paradigm of spectral calculations has been radically reversed – an interesting fact that has not yet obtained the necessary treatment in textbooks.

Conclusions. In this tutorial paper we disclose the main accuracy killers in computer calculations performed in floating-point FMA. In order of appearance they are: 1. *Over– and underflows*, 2. *Loss of left-most true digits*, 3. *Loss of right-most true digits*, 4. *High sensitivity in function evaluation* (*incl.* 4.1. *Large argument*, 4.2. *Large derivative of the evaluated function and* 4.3. *Small function value*), 5. *Intermediate results which are large compared to the final result*, 6. *Improper use of residuals in solving linear and non-linear finite equations* and 7. *Inappropriate decomposition of computational problems*. Only taking into account these and other possible destroyers of accuracy in the computed result, one may develop reliable numerical procedures in FMA.

The action of these factors is easily demonstrated by simple examples realized in MATLAB environment.

REFERENCES

[1] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comp. Surveys*, **23** (1991), No 1, 5–48, `http://www.validlab.com/goldberg/paper.pdf`.

[2] N. Higham. Accuracy and Stability of Numerical Algorithms. SIAM, Philadelphia, 2002.

[3] N. Higham, M. Konstantinov, V. Mehrmann, P. Petkov. The sensitivity of computational control problems. *IEEE Control Systems Magazine*, **24** (2004), 28–43 (PDF text available at `http://ieeexplore.ieee.org`).

[4] IEEE Standards for Binary Floating-Point Arithmetic. ANSI/IEEE Standard 754-1985, IEEE, New York, 1985 [Repr. in *SIGPLAN Not.*, **22** (1987), No 2, 9–25].

[5] IEEE Standard for Floating-Point Arithmetic 754-2008, IEEE, New York, 2008, ISBN 978-0-7381-5753-5,
http://ieeexplore.ieee.org/ xpl/mostRecentIssue.jsp?punumber=4610933
[6] M. Konstantinov, P. Petkov. Effects of finite arithmetic in numerical computations. In: Topics Contemp. Diff. Geometry, Complex Anal. and Math. Physics. World Sci., Singapore, 2007, 146–157.
[7] M. Konstantinov, P. Petkov, Z. Gancheva. Thirteen myths in numerical analysis. *Math. and Education in Math.*, **34** (2005), 237–242.
[8] C. Moler. Numerical Computing with MATLAB. The MathWorks, Inc., 2004 (PDF text available at http://www.mathworks.com/moler).
[9] J. Reiser, D. Knuth. Evaluating the drift in floating-point addition. *Inform. Proc. Lett.*, **3** (1975), No 3, 84–87.

Mihail M. Konstantinov
Department of Mathematics
University of Architecture, Civil Engineering and Geodesy
1046 Sofia, Bulgaria
e-mail: mmk_fte@uacg.bg

Petko H. Petkov
Department of Automatics
Technical University of Sofia
1756 Sofia, Bulgaria
e-mail: php@tu-sofia.bg

ЗАГУБА НА ТОЧНОСТ В ЧИСЛЕНИТЕ ПРЕСМЯТАНИЯ

Михаил М. Константинов, Петко Х. Петков

Разгледани са възможните катастрофални ефекти от неправилното използване на крайна машинна аритметика с плаваща точка. За съжаление, тази тема не винаги се разбира достатъчно добре от студентите по приложна и изчислителна математика, като положението в инженерните и икономическите специалности в никакъв случай не е по-добро. За преодоляване на този образователен пропуск тук сме разгледали главните виновници за загубата на точност при числените компютърни пресмятания. Надяваме се, че представените резултати ще помогнат на студентите и лекторите за по-добро разбиране и съответно за избягване на основните фактори, които могат да разрушат точността при компютърните числени пресмятания. Последното не е маловажно – числените катастрофи понякога стават истински, с големи щети и човешки жертви.