Article

Not peer-reviewed version

Brute Force Computations and Reference Solutions

Mihail Mihaylov Konstantinov , Petko Hristov Petkov * , Ekaterina Borisova Madamlieva

Posted Date: 9 December 2024

doi: 10.20944/preprints202412.0780.v1

Keywords: computational algorithms; brute force computations; reference solutions; machine arithmetic; failure of computer codes

 doi:10.20944/preprints202412.0780.v1

Article

Brute Force Computations and Reference Solutions

Mihail Konstantinov [1], Petko Petkov [2,*] and Ekaterina Madamlieva [3]

[1] University of Architecture, Civil Engineering and Geodesy, Sofia 1064 Bulgaria
[2] Bulgarian Academy of Sciences, Sofia 1040, Bulgaria
[3] Technical University of Sofia, 1000 Sofia, Bulgaria
* Correspondence: php@tu-sofia.bg

Abstract: In this paper, we consider the application of brute force computational techniques (BFCT) for solving computational problems in mathematical analysis and matrix algebra in floating-point computing environment. These techniques include, among others, simple matrix computations and analysis of graphs of functions. Since BFCT are based on matrix calculations the program system MATLAB® is suitable for their computer realization. The computations in this paper are done in double precision floating-point arithmetic obeying the 2019 IEEE Standard for binary floating-point calculations. One of the aims of the paper is to analyze cases when popular algorithms and software fail to produce correct answers without warning for the user. In real time control applications this may have catastrophic consequences with heavy material damage and human casualties. It is known, or suspected, that a number of man-made catastrophes such as Dharhan accident (1991), Ariane 5 launch failure (1996), Boeing 737 Max tragedies (2018, 2019) and others are due to errors in the computer software and hardware. Another application of BFCT is finding good initial guesses for known computational algorithms. Sometimes simple and fast BFCT are useful tools in solving computational problems correctly and in real time. Among particular problems considered are genuine addition of machine numbers, numerically stable computations, finding minimums of arrays, minimization of functions, solving finite equations, integration and differentiation, computing condensed and canonical forms of matrices and clarifying the concepts of least squares method in the light of the conflict Remainders vs. Errors. Usually BFCT are applied under user's supervision which is not be possible in automatic implementation of computational methods. To implement BFCT automatically is a challenging problem in the area of artificial intelligence (AI) and of mathematical artificial intelligence (MAI) in particular. BFCT allow to reveal the underlying arithmetic (FPA, VPA, etc.) in the performance of computational algorithms. Last but not least this paper may have some tutorial value since at certain places computational algorithms and mathematical software are still taught without taking into account the properties of computational algorithms and machine arithmetic.

Keywords: computational algorithms; brute force computations; reference solutions; machine arithmetic; failure of computer codes

MSC: 65-04; 65Y04

1. Introduction and Notation

1.1. Preliminaries

Sophisticated computational algorithms have been developed during the last 300 years for solving a large variety of problems in mathematical and engineering sciences [1–5]. When digital computers become available since the middle of 20th century it became clear that some computational algorithms of the past are not suitable for computer implementation at least without significant modifications. Other algorithms (albeit very clever) became obsolete. Thus paradigms of scientific computations had been changed and new types of modern computational algorithms were developed and implemented as computer codes.

When implemented as computer codes in floating-point computer environment such as MATLAB® [6,7] and Maple [8], however, some of these computational algorithms may produce wrong results without warning. Other systems as Mathematica [9,10] use variable precision arithmetic

 doi:10.20944/preprints202412.0780.v1

(VPA) and may avoid such pitfalls at the price of certain increase of the time necessary for performance of the algorithms. In such cases BFCT and a detailed analysis of computational problems with reference solution (CPRS) may be useful.

BFCT may also be used to obtain initial guesses for computational algorithms intended to solve finite equations and minimize functions of one or several scalar arguments. We stress finally that Monte Carlo algorithms [11] are also a form BFCT. As a perspective, a combination of mathematical AI [12,13] with BFCT is yet to be developed.

The computations in this paper are done in double precision binary floating-point arithmetic (FPA) obeying the IEEE Standard [14]. Some of the matrix notations used are inspired by the language of MATLAB®. The codes for the numerical experiments are given so the results may be checked independently. BFCT allow to reveal the underlying machine arithmetic.

1.2. General Notations

Next we use the following general notations.

- $\mathbb{Z}$ – set of integers
- $\mathbb{Z}[m,n]$ – set of integers $m, m+1, \dots, n$, where $m \le n$
- $\mathbb{Z}_k \subset \mathbb{Z}$ – set of integers $\ge k$
- $\mathbb{R}$ ($\mathbb{C}$) – set of real (complex) numbers
- $\mathrm{i} \in \mathbb{C}$ – imaginary unit, where $\mathrm{i}^2 = -1$
- $\mathrm{sum}(a_k, n)$ – sum of the scalar quantities $a_1, a_2, \dots, a_n$
- $\mathrm{int}(f, a, b)$ – definite integral of the function $f : \mathbb{R} \to \mathbb{R}$ on the interval $[a, b]$
- $\prec$ – lexicographical order on the set of pairs $(a, b) \in \mathbb{R}^2$ defined as $(a_1, b_1) \prec (a_2, b_2)$ if $a_1 < a_2$, or $a_1 = a_2$ and $b_1 < b_2$; we write $(a_1, b_1) \preceq (a_2, b_2)$ if $(a_1, b_1) \prec (a_2, b_2)$, or $(a_1, b_1) = (a_2, b_2)$
- $\mathbb{K}(m,n)$ – set of $m \times n$ matrices A with elements $A(p,q) \in \mathbb{K}$, where $\mathbb{K}$ is one of the sets $\mathbb{Z}$, $\mathbb{R}$ or $\mathbb{C}$
- $\mathbb{K}(m) = \mathbb{K}(m,m)$
- $A(p,:) \in \mathbb{K}(1,n)$ – pth row of $A \in \mathbb{K}(m,n)$
- $A(:,q) \in \mathbb{K}(m,1)$ – qth column of $A \in \mathbb{K}(m,n)$
- $\|\cdot\|$ – spectral norm in $\mathbb{K}(m,n)$
- $\mathrm{ones}(m,n) \in \mathbb{R}(m,n)$ – matrix with unit elements
- $\mathrm{rank}(A) \in \mathbb{Z}_0$ – rank of the matrix $A \in \mathbb{K}(m,n)$
- $\mathrm{tr}(A) \in \mathbb{K}$ – trace of the matrix $A \in \mathbb{K}(m,n)$
- $\det(A) \in \mathbb{K}$ – determinant of the matrix $A \in \mathbb{K}(m)$
- $I_n \in \mathbb{R}(n)$ – identity matrix with $I_n(p,p) = 1$ and $I_n(p,q) = 0$ for $p \ne q$
- $E_{i,j}$– matrix with unique nonzero element, equal to 1, in position (i,j)
- $N_n \in \mathbb{R}(n)$ – nilpotent matrix with $N_n(p,p+1) = 1$ and $N_n(p,q) = 0$ for $q \ne p+1$
- $\lambda_k = \lambda_k(A)$ – eigenvalues of the matrix $A \in \mathbb{K}(n)$, where $k = 1, 2, \dots, n$; we usually assume that $\lambda_1 \preceq \lambda_2 \preceq \cdots \preceq \lambda_n$
- $\sigma_k = \sigma_k(A)$ – singular values of the matrix $A \in \mathbb{K}(m,n)$, where $k = 1, 2, \dots, p$, $p = \min\{m,n\}$; the singular values are ordered as $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_p \ge 0$
- $\|A\| = \sigma_1$ – spectral norm of the matrix $A \in \mathbb{K}(m,n)$
- $\mathrm{cond}(A) = \|A\|\,\|A^{-1}\| = \sigma_1/\sigma_n \ge 1$ – condition number of the non-singular matrix $A \in \mathbb{K}(n)$

1.3. Machine Arithmetic Notations

We use the following notations from binary floating-point arithmetic, where `codes` and `results` in MATLAB®environment are written in `typewriter font`.

- $\mathbb{M}_+$ ($\mathbb{M}_-$) – set of finite positive (negative) machine numbers
- $\mathbb{M} = M_- \cup \{0\} \cup \mathbb{M}_+$ – set of finite machine numbers
- $\mathrm{fl}(x) \in \mathbb{M}$ – rounded value of $x \in \mathbb{R}$; $\mathrm{fl}(x) \in \mathbb{M}$ is computed as $m/2^n$ by the command `sym(x,'f')`, where $m \in \mathbb{Z}$, $n \in \mathbb{Z}_0$ and m is odd if $n \in \mathbb{Z}_1$
- $\mathbb{M}_+$ ($\mathbb{M}_-$) – set of positive (negative) machine numbers; we have $\mathbb{M} = \mathbb{M}_- \cup \{0\} \cup \mathbb{M}_+$

 doi:10.20944/preprints202412.0780.v1

- $X^+ > X$ ($X^- < X$) – machine number right (left) nearest to $X \in \mathbb{M}$; X^+ is obtained as `X + eps(X)`
- $\text{eps} = 2^{-52}$ – machine epsilon; we have $1^+ = 1 + \text{eps}$, $1^- = 1 - \text{eps}/2$; eps is obtained as `eps(1)`, or simply as `eps`
- $\rho = \text{eps}/2 = 2^{-53} \simeq 1.11 \times 10^{-16}$ – rounding unit
- $R = 2^{53} = 9\,007\,199\,254\,740\,992 \simeq 9.01 \times 10^{15}$ – maximal integer such that positive integers $\leq R$ are machine numbers; we have $R^+ = R + 2$, $R^- = R - 1$
- $E = 2^{1023}$ – maximal machine integer degree of 2; we have $2E \notin \mathbb{M}$
- $X_{\max} = (2 - \text{eps})\text{E} \simeq 1.80 \times 10^{308}$ – maximal machine number; it is also denoted as `realmax` and we have `eps(realmax) = 2^971`
- $X_{\min} = 2^{-1022} \simeq 2.23 \times 10^{-308}$ – minimal positive normal machine number; it is also denoted as `realmin`
- $R_{\min} = 2\text{eps}/\text{E} = 2^{-1074} \simeq 4.94 \times 10^{-324}$ – minimal positive subnormal machine number; it is found as `eps(0)`
- $\text{fl}(x) \in \mathbb{M}$ – rounded value of $x \in \mathbb{R}$
- $\mathcal{X} = [X_{\min}, X_{\max}] \subset \mathbb{R}$ – normal range; numbers $x \in \mathcal{X}$ are rounded with relative error less than ρ
- `Inf (-Inf)` – positive (negative) machine infinity
- `NaN` – result of mathematically undefined operation, comes from Not a Number
- $\mathbb{M}_{\text{ext}} = \mathbb{M} \cup \{\pm\text{Inf}\} \cup \{\text{NaN}\}$ – extended machine set

1.4. Abbreviations

AI	–	artificial intelligence
ARRE	–	average rounding relative error
BFCT	–	brute force computational technique
CPRS	–	computational problem with reference solution
FPA	–	double precision binary floating-point arithmetic
LLSP	–	linear least squares problem
LSM	–	least squares method
LSP	–	least squares problem
MAI	–	mathematical artificial intelligence
MRRE	–	maximal rounding relative error
NLF	–	Newton-Leibniz formula
NFP	–	numerical fixed point
RRE	–	rounding relative error
RTA	–	real time application
VPA	–	variable precision arithmetic

1.5. Software and Hardware

The computations are done with the program system MATLAB®, Version 9.9 (R2020b), on Lenovo X1 Carbon 7th Generation Ultrabook.

1.6. Problems with Reference Solution

An effective approach to using BFCT is solving CPRS. CPRS are specially designed computational problems with a priory known, or reference, solutions. It is supposed that the numerical algorithm which is to be checked does not "know" this fact and solves the computational problem in the standard way. Consider for example the finite equation $f(x) = 0$, where $f(x)$ and x are scalars or vectors. Then a CPRS is obtained choosing $f(x) = g(x) - g(x_{\text{ref}})$, where x_{ref} is the reference solution. We usually choose x_{ref} to be a vector with unit elements, e.g. `x_ref = ones(n,1)` and $\|x_{\text{ref}}\| = \sqrt{n}$.

Let $x_{\text{exact}} \neq 0$ be a solution of the equation $f(x) = 0$ and x_{comp} be an approximate solution computed in FPA. The absolute error $\|x_{\text{comp}} - x_{\text{exact}}\|$ and the relative error

$$\frac{\|x_{\text{comp}} - x_{\text{exact}}\|}{\|x_{\text{exact}}\|}$$

 doi:10.20944/preprints202412.0780.v1

of the computed solution x_{comp}, such that $f(x_{\text{comp}}) \simeq 0$, depends on rounding of initial data and on rounding and other errors made during the computational procedure [15,16]. To eliminate effects of rounding of initial data we may consider CPRS with relatively small integer data, or more generally, with data consisting of machine numbers.

Example 1. *Computational algorithms and software for solving linear algebraic equations $Ax = b$, where $A \in \mathbb{C}(n)$ is invertible, may be checked choosing $A \in \mathbb{Z}(n)$ and $b = Ax_{\text{ref}}$, where* `x_ref = ones(n,1)`.

The computed solution is `x_comp = A\b` *and we may compare the relative error* `norm(x_ref-x_comp)/sqrt(n)` *with the a priori error estimate* `eps*cond(A)`.

Example 2. *Let $F : [a,b] \to \mathbb{R}$ be a differentiable highly oscillating function with derivative $F' : [a,b] \to \mathbb{R}$. The numerical integration of $f = F'$ may be a problem. For such integrands we may formulate a numerically difficult computational problem $J = \text{int}(f,a,b)$ with reference solution $J_{\text{ref}} = 1$. Systems with mathematical intelligence such as MATLAB® [6], Maple [8] and Mathematica [9,10] may find the integral with high accuracy by the NLF $J = F(b) - F(a)$ thus avoiding numerical integration.*

Example 3. *To check the codes for eigenstructure analysis of a matrix $A \in \mathbb{C}(n)$ we may choose $A = T(I_n + N_n)T^{-1}$, where T, T^{-1} are integer matrices with $\det(T) = 1$. The eigenvalues of A are equal to 1 (the reference solution). Computational algorithms such as the QR algorithm and software for spectral matrix analysis may produce wrong results even for matrices $A \in \mathbb{Z}(2)$ of moderate norm.*

Example 4. *When solving the differential equation $y'(t) = f(t, y(t))$ with initial condition $y(t_0) = y_0$ we may choose a reference solution y_{ref} as follows. The function y_{ref} satisfies the differential equation $y'(t) = \Phi(t, y(t))$, where*

$$\Phi(t, y(t)) = f(t, y(t)) + y'_{\text{ref}}(t) - f(t, y_{\text{ref}}(t))$$

with initial condition $y_{\text{ref}}(t_0) = y_0$.

An important observation is that for some computational problems such as finding extrema of functions, BFCT may produce better results than certain sophisticated algorithms used in modern computer systems for doing mathematics, see e.g. Section 7.

2. Machine Arithmetic

2.1. Preliminaries

In this section we consider computations in FPA obeying the IEEE Standard [14]. Numerical computations in MATLAB® are done according to this standard. The FPA consists of set of machine numbers $\mathbb{M}$ and rules for performing operations, e.g. addition/subtraction, multiplication/division, exponentiation, etc. Rounding and arithmetic operations in FPA are performed with RRE of order ρ. The sum $A + B + C$ and the product ABC are computed as $(A + B) + C$ and $(AB)C$.

A major source of errors in computations is the (catastrophic) cancellation when relatively exact close numbers are subtracted so that the information coded in their left-most digits is lost [17]. Thus catastrophic cancellation is the phenomenon when subtracting good approximations to close numbers may cause a bad approximation to their difference. Less known is that genuine addition in FPA may also cause large and even unlimited errors.

A tutorial example of cancellation is solving quadratic equations $ax^2 + bx + c = 0$, $a \neq 0$. During the last 4000 years students are taught to use the formula $x_{1,2} = (-b \pm \sqrt{d})/(2a)$, $d = b^2 - 4ac$, without warning that this may be a bad way to solve numerically quadratic equations.

Similar considerations are valid for the use of explicit expressions for solving cubic and quartic algebraic equations. Because genuine subtraction is numerically dangerous. Genuine addition in FPA is not less dangerous.

 doi:10.20944/preprints202412.0780.v1

2.2. Violation of Arithmetic Laws

In FPA the commutative law for addition and multiplication is valid but the associative law for addition and multiplication is violated, i.e. it may happen that

$$(A+B)+C \neq A+(B+C),\ (AB)C \neq A(BC)$$

The distributive law $A(B+C) = AB+AC$ is also violated. This may lead to unexpectedly large (in fact arbitrarily large) errors.

For example, we have the wrong result $1+R-R = (1+R)-R = R-R = 0$. The reason for this machine calculation is that $1+R$ is not a machine number (this is the smallest positive integer with this property). It lies in the middle of the successive machine numbers R and $2+R$ and is rounded to R. At the same time we have $1+(R-R) = 1+0 = 1$ which is the correct answer. Thus we got the famous wrong equality $\boxed{0=1}$ arising in mathematical jokes.

This is not the end of the story. Consider the expression $S = R+1+1+\cdots+1-R$ consisting of $2+R$ members, where there are R members equal to 1. We have $S = R$. Computing S from left to right we get the wrong answer $S = (R+1+1+\cdots+1)-R = R-R = 0$. Computing S starting with the ones, i.e. $1+1+\cdots+1+R-R = R+R-R$ we get the correct answer $S = R$. Now we have obtained the more impressive result $\boxed{0 = 9\,007\,199\,254\,740\,992}$. Summing (at least theoretically) sufficient number of units we compute the sum of 1's as `Inf` and hence $\boxed{0=\text{Inf}}$.

The associative law for multiplication is also violated in FPA. For example, the value of the expression $P = 2^{512}\times 2^{512}\times 2^{-512}$ is 2^{512} and it is computed correctly as

$$P = 2^{512}\times\left(2^{512}\times 2^{-512}\right) = 2^{512}\times 1 = 2^{512}$$

At the same time the machine computation of P from left to right gives

$$\left(2^{512}\times 2^{512}\right)\times 2^{-512} = \text{fl}(2^{1024})\times 2^{-512} = \text{Inf}\times 2^{-512} = \text{Inf}$$

where Inf is the symbol for infinity in FPA. Thus we got $\boxed{2^{512}=\text{Inf}}$.

Similar false equalities may also be obtained by addition and subtraction of machine numbers. Set $E = 2^{1023}$. In standard arithmetic we have $E+E-E-E = 0$. In FPA it is fulfilled $\text{fl}(E+E-E-E) = \text{fl}(E+E)-E-E = \text{Inf}-E-E = \text{Inf}$. This corresponds to the wrong equality $\boxed{0=\text{Inf}}$ although E is a machine number at a good distance $E-2^{971}$ to $X_{\max}$.

We also have $\text{fl}(E+E-E-E-E-E) = \text{fl}(E+E)-E-E-E-E = \text{Inf}-E-E-E-E = \text{Inf}$ although the exact result is $-2E$ and should be rounded to `-Inf`. Thus we obtained $\boxed{-\text{Inf}=\text{Inf}}$. The reason is that the maximum positive machine number in FPA is $X_{\max} = (2-\text{eps})\text{E}$ and the number $E+E = 2^{1024}$ is set to Inf. Moreover, the maximum positive integer that is still rounded to $X_{\max}$ is $X_{\max}+2^{969}(2-\text{eps})$ while the number $X_{\max}+2^{970}$ is set to `Inf`.

These entertaining exercises may produce undetermined results as well. For example, we have $U = 2^{1024}/2^{1024}-1 = 0$ but the computed value is `U = Inf/Inf - 1 = NaN`. This may be interpreted as the exotic equality $\boxed{0=\text{NaN}}$.

Working with positive numbers $x \leq X_{\min}$ also leads to violation of the distributive law $(x+y)z = xz+yz$. In particular, we usually think that for $x\in\mathbb{R}$ it is fulfilled $(x+x)/2 = x/2+x/2 = x$. But setting $x = X_{\min}$ we get `(x + x)/2 = 2*x/2 = x` and `x/2 + x/2 = 0 + 0 = 0`. We stress that the minimal positive quantity that is still rounded to $X_{\min}$ instead to 0 is $X_{\min}/(2-\text{eps})$.

The distributive law in the form `(1/m + 1/n)/p = 1/m/p + 1/n/p`, where at least one of the operands $m,n,p\in\mathbb{Z}$ is not an integer degree of 2, is also violated. In this case the rounded values of both sides of the equality are usually neighboring machine numbers.

 doi:10.20944/preprints202412.0780.v1

Example 5. *Let* $m = 2$, $n = 3$, $p = 5$. *Then* $\mathrm{fl}(1/2/5 + 1/3/5) = \mathrm{fl}(1/6) + 2^{-55}$ *and* $\mathrm{fl}(1/2/5 - 1/3/5) = \mathrm{fl}(1/30) + 2^{-57}$.

Example 5 illustrates the fact that machine subtraction of close numbers a, b is usually performed with small relative error. Moreover, if the operands are machine numbers and $a/2 \leq b \leq 2a$ then the machine subtraction is exact [17].

The rounded value $\mathrm{fl}(X) \in \mathbb{M}$ of $X \in \mathbb{R}$ is computed by the command `sym(X,'f')` in the form $\pm m/2^n$, where $m, n \in \mathbb{Z}_0$ and m is odd when $n \geq 1$. In particular the command `sym(realmax,'f')` will give 309 decimal digits of the integer $X_{\max}$ = `realmax`.

2.3. Numerical Convergence of Series

We shall conclude our brief excursion in the machine summation of numbers with the consideration of numerical series with positive elements a_k. For $n \in \mathbb{Z}_1$ set $\mathrm{sum}(a_k, n) = a_1 + a_2 + \cdots + a_n$.

Definition 1. *The symbol* $\mathrm{sum}(a_k, \infty)$ *is called numerical series. The quantity* $A_n = \mathrm{sum}(a_k, n)$*is the n-th partial sum of this series.*

The partial sums are computed in FPA as $A_1 = a_1$, $A_{m+1} = \mathrm{fl}(a_m + A_m) \in \mathbb{M}$, $m \in \mathbb{Z}_1$. Theoretically, the series $\mathrm{sum}(a_k, \infty)$ is divergent when $a_k \geq X_{\min}$. In FPA there are three mutually disjoint possibilities according to the next definition.

Definition 2. *The series* $\mathrm{sum}(a_k, \infty)$ *is said to be:*

1. *numerically convergent if there is a positive* $S \in \mathbb{M}$ *and* $m \geq 1$ *such that* $A_n = S$ *for* $n \geq m$*; the number* S *is called numerical sum of the series* $\mathrm{sum}(a_k, \infty)$.
2. *numerically divergent if there is* $m > 1$ *such that* $A_n = \mathrm{Inf}$ *for* $n \geq m$*; in this there are now terms* $A_n = \mathrm{NaN}$ *for* $n \geq m$
3. *numerically undefined if there is* $m > 1$ *such that* $A_n \neq \mathrm{NaN}$ *for* $n \leq m$ *and* $A_{n+1} = \mathrm{NaN}$

Divergent numerical series may be (and very often are!) numerically convergent. For example, the divergent harmonic series with $a_k = 1/k$ is numerically convergent in FPA to a sum $S \simeq 36$. The divergent series with $a_k = 1$ is numerically convergent to $S = R$. Also, it is easy to prove the following assertion.

Proposition 1. *For any* $S \in \mathbb{M}_+$ *there is a series* $\mathrm{sum}(a_k, \infty)$ *which is numerically convergent to* S.

Proposition 1 for FPA is an analogue to the Riemann series theorem [18] which states that the members of a convergent real series which is not absolutely convergent may be reordered so that the sum of the new series to be equal to any prescribed real number.

2.4. Average Rounding Errors

We recall that the normal range of FPA is the interval $\mathcal{X} = [R_{\min}, X_{\max}] \subset \mathbb{R}$, where numbers are rounded with RRE less than ρ.

Let $x \in \mathcal{X}$ and $\mathrm{fl}(x) \in \mathbb{M}$ be the rounded value of x. It is well known [14] that the RRE satisfies the estimate

$$\mathrm{rre}(x) = \frac{|x - \mathrm{fl}(x)|}{|x|} \leq \frac{\rho}{1+\rho} < \rho \tag{1}$$

The code `sym(x,'f')` computes $\mathrm{fl}(x)$ in the form $m/2^n$, where m is odd if $n \geq 1$. Let $n \in \mathbb{Z}[0, 2^{52} - 1]$. The expression $\mathrm{rre}(x)$ between two successive machine numbers

$$X_n = 1 + 2n\rho,\ X_{n+1} = 1 + 2(n+1)\rho$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 December 2024 doi:10.20944/preprints202412.0780.v1

in the interval $[1,2)$ is as follows. Let

$$x_n = \frac{1}{2}(X_n + X_{n+1}) = 1 + (2n+1)\rho \notin \mathbb{M}$$

Then

$$\mathrm{rre}(x) = \begin{cases} 1 - X_n/x & X_n \le x \le x_n \\ X_{n+1}/x - 1 & x_n < x \le X_{n+1} \end{cases}$$

The maximum of $\mathrm{rre}(x)$ in the interval $[X_n, X_{n+1}]$ is achieved for $x = x_n$ and is equal to

$$e_n = \mathrm{rre}(x_n) = \frac{X_{n+1} - X_n}{X_n + X_{n+1}} = \frac{\rho}{1 + (2n+1)\rho}$$

The graph of the function $\mathrm{rre}\colon [1,2) \to \mathbb{R}$ resembles a saw, see Figure 1.

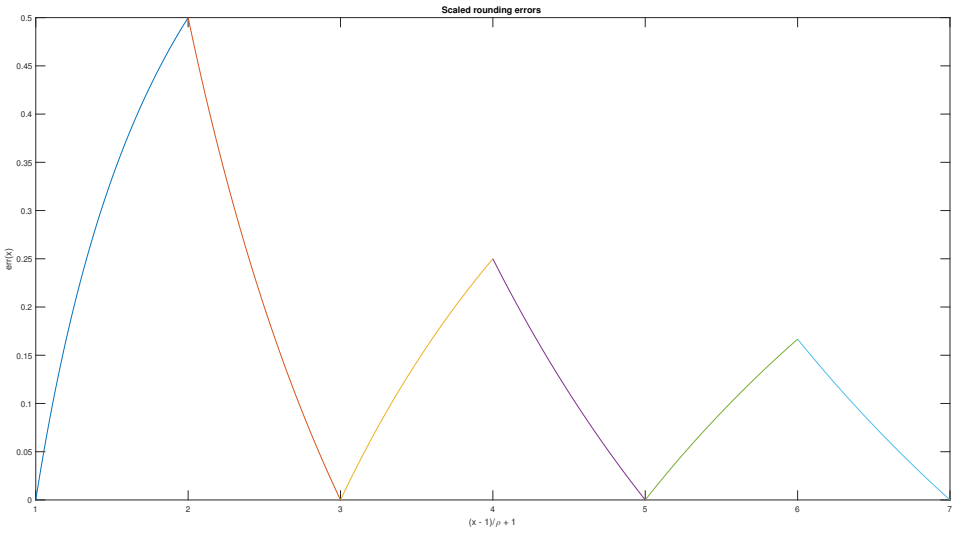

Figure 1. Scaled rounding errors.

For $x > X_n$ near to X_n the maximum RRE is about ρ, while for $x < X_{n+1}$ near to X_{n+1} the maximum RRE is close to $\rho/2$. In particular we have

$$\begin{aligned} \mathrm{rre}(1+\rho) &= \frac{\rho}{1+\rho} = \rho - \rho^2 + \mathrm{O}(\rho^3) \\ \mathrm{rre}(2-2\rho) &= \frac{\rho}{2(1-\rho)} = \frac{1}{2}\rho + \frac{1}{2}\rho^2 + \mathrm{O}(\rho^3) \end{aligned} \tag{2}$$

That actual behavior of RRE is not governed by (1) but is considerably smaller, has been known to specialists in computer calculations from a long time. To clarify this problem we performed intensive numerical experiments.

BFCT showed that actual RRE is considerably lower than ρ and lower than $\rho/2$ and this led to the definition and use of average RRE which are more realistic in comparison with the relative error ρ. Based on this observation [19] we have defined and calculated an integral ARRE as the definite integral $\mathrm{int}(\mathrm{rre}, 1, 2)$. This integral is of order

$$K\rho + \mathrm{O}(\rho^2),\ K = \frac{1}{2}\log(2) \simeq 0.3466 \tag{3}$$

 doi:10.20944/preprints202412.0780.v1

At the same time the average of the maximums of RRE is

$$2K\rho = \log(2)\rho \simeq 0.6931\,\rho.$$

Intensive numerical BFCT experiments [19] with rounding of large sequences of numbers $x \notin \mathbb{M}$ had shown that the actual ARRE is about $0.40\,\rho$. This is 2.5 times less than the widely used bound ρ in the literature [14]. In these experiments numbers $x \in \mathbb{M}$ had been excluded because $\mathrm{rre}(x) = 0$. This explains why the multiplier of ρ was computed as 0.40 instead of 0.35.

In another series of experiments we have computed and averaged the RRE for fractions $x = 1/k$, $x = 1 + 1/k$ and $x = 2 - 1/k$ for $k = 1\colon m$ and m up to 1000 and we have computed the ratios $\mathrm{rre}(x)/\rho$. The results are shown in Table 1. The actual RRE in rounding of unit fraction $1/m$ (second column) is very close to the integral ARRE (3).

For fractions $1 + 1/m$ close to 1 the computed ARRE is larger than for fractions $2 - 1/m$ close to 2. This confirms the observation that rounding of $x \in [2^k, 2^{k+1}]$, $k \in \mathbb{Z}$, is done with less (although not twice less) RRE for x close to the left limit 2^k than close to the right limit 2^{k+1} of the interval, see (2). For example, the result in field (4,2) of Table 1 is obtained by the code

```
for m = 1:1000
 R(m) = double(abs((sym(1/m,'e')-1/m))*m*2/eps);
end
mean(R)
```

Table 1. Ratios $\mathrm{rre}(x)/\rho$ for certain fractions.

m	$1/m$	$1+1/m$	$2-1/m$
10	0.3000	0.2423	0.1950
100	0.3417	0.2443	0.2413
1000	0.3323	0.2649	0.2316

We stress that in such experiments we cannot use pseudo-random number generators such as `rand`, `randn` and `randi` from MATLAB® since they produce machine numbers for which the rounding errors are zero.

We summarize the above considerations in the next important proposition.

Proposition 2. *The known rounding error bounds in all computational algorithms realized in FPA may be reduced three times (!) if integral ARRE are used instead of maximum RRE.*

3. Numerically Stable Computations

3.1. Lipschitz Continuous Problems

Numerical stability of computational procedures is a major issue in numerical analysis [15]. A large variety of computational problems (actually, almost all computational problems) may be formulated as follows, see [15,16]. Let $\mathcal{D} \subset \mathbb{C}(m,1)$ and $\mathcal{R} \subset \mathbb{C}(n,1)$ be given sets and let $f\colon \mathcal{D} \to \mathcal{R}$ be a continuous function. We shall use the following informal definition.

Definition 3. *The computational problem is described by the function f, while the particular way of computing the result $r = f(d)$ for a given data $d \in \mathcal{D}$ is the computational algorithm.*

Thus the computational problem is identified with the pair (f, d), or with the equality $r = f(d)$. For many computational problems the function f is Lipschitz continuous.

 doi:10.20944/preprints202412.0780.v1

Definition 4. *The computational problem $r = f(d)$ is said to be (locally) Lipschitz continuous if there exist constants $L > 0$ and $b > 0$ such that*

$$\|f(d_1) - f(d_2)\| \leq L\|d_1 - d_2\|$$

whenever $\|d_1\|, \|d_2\| \leq b$. The problem is globally Lipschitz continuous if $b = \infty$.

If the function f has locally bounded derivative it is (locally) Lipschitz continuous. The concept of Lipschitz continuity is illustrated by the next example.

Example 6. *Let $f : \mathcal{D} \to \mathbb{R}$, $\mathcal{D} \subset \mathbb{R}$, and $a > 0$ be a given constant. Then the following assertions take place.*

1. *The function $f(d) = 1/d$ is Lipschitz continuous if $\mathcal{D} = (-\infty, a] \cup [a, \infty)$ and is not Lipschitz continuous if $\mathcal{D} = \mathbb{R}\backslash\{0\}$.*
2. *The function $f(d) = d^{1/3}$ is Lipschitz continuous if $\mathcal{D} = (-\infty, a] \cup [a, \infty)$ and is not Lipschitz continuous if $\mathcal{D} = \mathbb{R}\backslash\{0\}$.*
3. *The function $f(d) = d^m$, $m \in \mathbb{Z}_2$, is Lipschitz continuous if $\mathcal{D} = [-a, a]$ and is not Lipschitz continuous if $\mathcal{D} = \mathbb{R}$.*
4. *The function $f(d) = d^{4/3}\sin(1/d)$ for $d \neq 0$ and $f(0) = 0$, is differentiable on $\mathbb{R}$ but is not Lipschitz continuous.*

Suppose finally that the algorithm is realized in FPA, e.g. in MATLAB® computing environment.

Definition 5. *The computational algorithm $r = f(d)$ is said to be numerically stable if the computed result r_{comp} is close to the result $f(d^*)$ of a close problem with data d^*.*

Definition 5 includes the popular concepts of forward and backwards numerical stability. To make this definition formal, suppose that there exist non-negative constants A, B such that

$$\|r_{\text{comp}} - f(d^*)\| \leq \rho A\|r\|, \ \|d - d^*\| \leq \rho B\|d\|$$

and $d \neq 0$, $r \neq 0$. Further on we have

$$\begin{aligned} \|r_{\text{comp}} - r\| &= \|r_{\text{comp}} - f(d^*) + f(d^*) - f(d)\| \\ &\leq \|r_{\text{comp}} - f(d^*)\| + \|f(d^*) - f(d)\| \\ &\leq \rho A\|r\| + L\|d^* - d\| \leq \rho A\|r\| + \rho BL\|d\| \end{aligned}$$

Dividing the last inequality by $\|r\|$ we get the following estimate for the relative error in the computed solution

$$\frac{\|r_{\text{comp}} - r\|}{\|r\|} \leq \rho C, \ C = A + \frac{BL\|d\|}{\|f(d)\|} \tag{4}$$

Definition 6. *The quantity $C = C(A, B, d)$ is said to be relative condition number of the computational problem $r = f(d)$.*

The remarkable formula (4) reveals the three main factors which determine the accuracy of the computed solution [16,19].

Proposition 3. *The accuracy of the computed solution r_{comp} depends on the following factors.*

1. *The sensitivity of the computational problem relative to perturbations in the data d measured by the Lipschitz constant L of the function f*
2. *The stability of the computational algorithm expressed by the constants A, B*

 doi:10.20944/preprints202412.0780.v1

3. *The FPA characterized by the rounding unit ρ and the requirement that the intermediate computed results belong to the normal range $\mathcal{X}$*

The error estimate (4) is used in practice as follows. For a number of problems the Lipschitz constant L may be calculated, or estimated as in the solution of linear algebraic equations and the computation of the eigenstructure of a matrix with simple eigenvalues. Finally, the value of ρ is known exactly for FPA as well as for other machine environments.

The estimate of the constants A, B may be a problem. Often the heuristic assumption $A = 0$ and $B = 1$ is made which gives

$$C = C(d) = \frac{L\|d\|}{\|f(d)\|} \tag{5}$$

It follows from (5) that C will be large if L and/or $\|d\|$ are large and/or $\|f(d)\|$ is small. The constant L is usually out user's control. The quantities $\|d\|$ and $\|f(d)\|$ may be changed by scaling of the computational problem

Definition 7. *The computational problem is said to be:*

1. *well conditioned if $\rho C \ll 1$; in this case we may expect about $-\log_{10}(\rho C)$ true decimal digits in the computed solution r_{comp}*
2. *poorly conditioned if $\rho C \simeq 1$; in this case there may be no true digits in the computed solution r_{comp}*

More precise classification of the conditioning (well, medium and poor) of computational problems is also possible. Computational problems which are not Lipschitz but Hölder continuous may also be analyzed, see e.g. [19].

The above considerations confirm a fundamental rule in numerical computations, namely that if the data d and/or some of the intermediate results in the computation of $f(d)$ are large and/or the result $r = f(d)$ is small then large relative errors in the computed solution r_{comp} may be expected, see Section 3.3.

For some computational problems $r = f(d)$ there are a priory error estimates for the relative error $\|r_{\text{comp}} - r\|/\|r\|$ in the computed solution r_{comp} in the form of explicit expressions in ρ and d. The importance of such estimates is that they may be computed (or estimated) a priory, before solving the computational problem.

Most a priory error estimates are heuristic. Among computational problems with such estimates is the solution of linear algebraic equations $Ax = b$ and the computation of the eigenvalues of a matrix A with simple spectrum. To check the accuracy and practical usefulness of such error estimates by BFCT, a set of CPRS is designed and the error estimates are compared with the observed errors.

3.2. Hölder Continuous Problems

Definition 8. *The computational problem $r = f(d)$ is said to be Hölder continuous with exponent $h > 0$ if there exist constants $L > 0$ and $b > 0$ such that*

$$\|f(d_1) - f(d_2)\| \leq L\|d_1 - d_2\|^h$$

whenever $\|d_1\|, \|d_2\| \leq b$.

Hölder continuity implies uniform continuity but the converse is not true. It is supposed that $h < 1$. Indeed, if $h = 1$ the problem is Lipschitz continuous and if $h > 1$ the function f is constant.

The machine computation of $r = f(d)$ may be accomplished with large errors of order eps^h when the exponent $h > 0$ is small. Such cases arise in the calculation of a multiple zero d_0 of a smooth function $f : \mathbb{R} \to \mathbb{R}$, where $f^{(k)}(d_0) = 0, k \in \mathbb{Z}[0, m-1]$ and $f^{(m)}(r_0) \neq 0$. In this case $h = 1/m$.

 doi:10.20944/preprints202412.0780.v1

Let $r \neq 0$ and r_{comp} be the computed value of r. Then a heuristic accuracy estimate for the solution of a Hölder problem is

$$\frac{\|r_{\text{comp}} - r\|}{\|r\|} \leq H\rho^h, \; H = L\frac{\|d\|^h}{\|r\|}$$

Experimental results confirming this estimate are presented later on.

A typical example of a Hölder problem is the computation of the roots of n-degree algebraic equation with multiple roots, where $n \geq 3$ (the case $n = 2$ is treated by a specific algorithm). There are three codes in MATLAB® that may be used for solving algebraic equations which may be characterized as follows.

1. The code `roots` is fast and works, on middle power computer platforms, with equations of degree n up to several thousands but gives large errors in case of multiple roots. This code solves algebraic equations only.
2. The code `vpasolve` works with VPA corresponding to 32 true decimal digits with equations of degree n up to several hundreds but may be slow in some cases. This code works with general equations as well finding one root at a time.
3. The code `fzero` is fast but finds one root at a time. It may not work properly in case of multiple roots of even multiplicity. This code works with general finite equations as well.

3.3. Golden Rule of Computations

The results presented in this and previous sections confirm the Golden Rule in doing computations in both exact and machine arithmetic which may be formulated as follows.

Proposition 4. *(**Golden Rule of Computations**) If in a chain of numerical computations the final result is small relative to the initial data and/or to the intermediate results then large relative errors in the computed solution are to be expected.*

The already mentioned catastrophic cancellation in subtraction of close numbers is a particular case of Proposition 4.

An important class of computations in FPA are the so called computations with maximum accuracy. Let the vector r with elements r_k be the exact solution of the vector computational problem $r = f(d)$ and let the vector r_{comp} with elements $r_k^* \in \mathbb{M}$ be the computed solution.

Definition 9. *The vector r_{comp} is computed with maximum accuracy if its elements r_k^* satisfy $r_k^* \in \{r_k^-, r_k^+\}$.*

Solutions with maximum accuracy are the best that we may expect from a computational algorithm implemented in FPA.

4. Extremal Elements of Arrays

Finding minimal and maximal elements of real and complex arrays is a part of many computational algorithms. This problem may look simple but it has pitfalls in some cases.

4.1. Vectors

Let $w = [w_1, w_2, \ldots, w_n]$ be a given real n-vector. The problem of determining an extremal (minimal or maximal) element w_k of w together with its index k arises as part of many computational problems. The solution of this problem is obtained in MATLAB® by the codes `[U,K] = min(w)` or `[V,L] = max(w)`, where $U = w_K$ is the minimal element of the vector w and K is its index. Similarly, $V = w_L$ is the maximal element of the vector w and L is its index. These codes work for complex vectors as well if the complex numbers are ordered lexicographically by the relation $\preceq$, see [6].

An important rule here is that if there are several elements w_k of w equal to its minimal element U then the computed index K is supposed to be the minimal one. Similarly, if there are several elements

 doi:10.20944/preprints202412.0780.v1

w_l of w equal to its maximal element V then the computed index L is again the minimal one. The delicate point here is the calculation of indexes K and L. For example, for $w = [c, c, c]$, where $c \in \mathbb{R}$ or $c \in \mathbb{C}$, the codes `min` and `max` produce $U = c$, $K = 1$ and $V = c$, $L = 1$ as expected.

Due to rounding the above codes may produce unexpected results. Indeed, the performance of the codes depends on the form in which the elements of the vector w are written.

Example 7. *The quantities $A = 1/3$, $B = 4/3 - 1$ and $C = 1/2 - 1/6$ are equal but their rounded values are different. The rounded value $\mathrm{fl}(X) \in \mathbb{M}$ of $X \in \mathbb{R}$ is computed by the code* `sym(X,'f')`. *It produces the answer in the form $m\,2^{-n}$, where $m, n \in \mathbb{Z}_0$. We have $\mathrm{fl}(A) = 6004799503160661/2^{54}$ and $\mathrm{fl}(B) = \mathrm{fl}(A) - 2^{-54}$, $\mathrm{fl}(C) = \mathrm{fl}(A) + 2^{-54}$. Hence $\mathrm{fl}(B) < \mathrm{fl}(A) < \mathrm{fl}(C)$.*

Example 8. *Let w be a 3-vector with elements A, B, C at any positions, where $A = B = C$ are defined in Example 7. Then the code* `[U,K] = min(w)` *will give $U = \mathrm{fl}(B)$ and K will be the index of B as an element of w. The code* `[V,L] = max(w)` *will give $V = \mathrm{fl}(C)$ and L will be the index of C as an element of w. At the same time both codes are supposed to give $K = L = 1$ and $U = V$.*

These details of the performance of the codes `min` and `max` may not be very popular even among experienced users of sophisticated computer systems for doing mathematics such as MATLAB®.

4.2. Matrices

Similar problems as in Section 4.1 arise in finding extremal (minimal or maximal) elements of multidimensional arrays and in particular of real and complex matrices A.

Consider a matrix $A \in \mathbb{C}(m, n)$ with elements $A(p, q)$. Let the minimal element of A relative to the lexicographical order be $A_{\min} = A(i_1, i_2)$. Then the code

```
[a,b] = min(A); [A_min,N] = min(a); A_min, ind = [b(N),N]
```

finds the minimal element $A_{\min}$ of A and the pair (i_1, i_2) of its indexes as

```
A_min, ind = i_1  i_2
```

where `i_1 = ind(1), i_2 = ind(2)`. Here a and b are n-vectors and `ind` is a 2-vector.

If there is more than one minimal element then the computed pair (i_1, i_2) of its indexes is minimal relative to the lexicographical order. This result, however, may depend on the way the elements of A are specified.

Finding minimal elements of matrices may be a part of improved algorithms to compute extrema of functions of two variables, see Section 7.2. A scheme for finding minimal elements of nD-arrays, $n > 2$, may also be derived and applied to the minimization of functions of n variables.

4.3. Application to Voting Theory

The above details in using the codes `min` and `max` from MATLAB® [6], although usually neglected, may be important. For example, in certain *automatic voting computations* (in which one of the authors of this paper had participated back in 2013) with the Hamilton method for the bi-proportional voting system used in Bulgaria, these phenomena actually occurred and caused problems. To resolve quickly the problem we applied BFCT using hand calculations (!) being in emergency situation.

The computational algorithms for realization of the Hamilton method must be improved replacing the fractional remainders by integer remainders [19] as follows. Let n parties with votes $v_1, v_2, \ldots, v_n$ take part in the distribution of $S > n$ parliamentary seats by the Hamilton method [20]. Set $V = v_1 + v_2 + \cdots + v_n$ and $m_k = Sv_k/V$.

Next the rationals m_k are represented as $\mathrm{fix}(m_k) + \mu_k/V$, where $\mathrm{fix}(x) \in \mathbb{Z}_0$ is the integer part of m_k, μ_k/V is the fractional remainder and $\mu_k \in \mathbb{Z}[1, V-1]$ is the integer remainder of m_k modulo V. Initially the kth party gets $\mathrm{fix}(m_k)$ seats. If the sum S_0 of initial seats is less than S then the first

 doi:10.20944/preprints202412.0780.v1

$S - S_0$ parties with largest integer remainders μ_k get one seat more. Thus to avoid small errors in the computation of fractional remainders that may cheat the code `max` we must work with exactly computed integer remainders.

5. Problems with Reference Solution

5.1. Evaluation of Functions

Evaluation of functions f defined by an explicit expression

$$y = f(x),\ 0 \neq x \in \mathbb{R},\ 0 \neq y \in \mathbb{R}$$

is a CPRS since $f(x)$ may be found with high precision in a suitable computing environment. However, the relative error in computing y may be large even for well behaved functions f and arguments x far from the limits of the normal range [19]. The reason is that instead with the argument $x \in \mathbb{R}$ the corresponding computational algorithm works with the rounded value $x^* = \mathrm{fl}(x) \in \mathbb{M}$. At the same time for $x \in \mathbb{M}$ the computed quantity $f(x^*)$ is usually exact to working precision.

Example 9. *We have*

```
[cos(realmax);sin(realmax)] = -0.999987689426560
                               0.004961954789184
```

The computed result is correct to full precision although the argument `realmax` *is written with 309 decimal digits, which may be displayed by the command* `sym(realmax,'f')`.

Let the function f be locally Lipschitz continuous with constant $L(x)$ in a neighborhood of x and let the result computed in FPA be $y^* = f(x^*)$. Then the relative error of y^* is estimated as

$$\frac{|y^* - y|}{|y|} \leq \rho C,\ C = C(f,x) = \frac{L(x)|x|}{|f(x)|}$$

where $C(f,x)$ is the relative condition number of the computational problem $y = f(x)$.

Example 10. *Consider the vector* $y = [\cos(2k\pi); \sin(2k\pi)]$, *where* $k \in \mathbb{Z}$. *We should have* $y = [1;0]$ *but for large* k *this may not be so. Let the vector* $X \in \mathbb{R}(1,8)$ *has elements* $X(p) = \pi\, 2^{46+p}$, *i.e.*

```
X = pi*[2^47,2^48,2^49,2^50,2^51,2^52,2^53,2^54]
```

The command `Y = [cos(X);sin(X)]` *gives the matrix*

```
Y = 0.9999  0.9994  0.9976  0.9905  0.9622  0.8517  0.4509 -0.5934
    0.0172 -0.0345 -0.0689 -0.1374 -0.2723 -0.5240 -0.8926 -0.8049
```

While the first column `Y(:,1)` *of* `Y` *still resembles the exact vector* `[1;0]`, *the next columns are heavily wrong with* `Y(:,8)` *being a complete catastrophe with relative error 170%. The commands* `Y1 = [C1;S1]` *and* `Y2 = [C2;S2]`, *where* `C1 = vpa(cos(X)), S1 = vpa(sin(X)), C2 = cos(vpa(X))` *and* `S2 = sin(vpa(X))` *produce the same wrong result* `Y1 = Y2 = Y`.

A way out from such wrong calculations is to evaluate standard trigonometric functions for arguments modulo 2π.

Example 11. *The command*

```
YY = [cos(mod(X,2*pi));sin(mod(X,2*pi))]
```

produces the matrix

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 December 2024 doi:10.20944/preprints202412.0780.v1


```
YY =  1     1     1     1     1     1     1     1
      0     0     0     0     0     0     0     0
```

which is exact to full precision.

We recall that for $|x| \geq R$ the rounded value $\mathrm{fl}(x)$ of x is an even integer. For example, the quantity $a = 23 * \pi^{32}/20$ is rounded to $\mathrm{fl}(a) = 9\,321\,670\,908\,397\,306$ which is demonstrated by the command `sym(23*pi^32/20,'f')`. Another instructive example is rounding integer degrees 10^n.

Example 12. *We have* $\mathrm{fl}(10^n) = 10^n$ *for* $n \in \mathbb{Z}[16, 22]$ *but*

```
sym([10^23;10^24;10^25],'f') =  99999999999999991611392
                                99999999999999983222784
                               100000000000000000905969664
```

If $x \in \mathbb{M}$ then for most elementary and special functions f the quantity $y = f(x)$ is computed with maximum accuracy.

If the function f is locally Lipschitz with constant $L(x)$ then large relative computational errors in evaluation of $y = f(x) \neq 0$ may occur when $C(f, x)$ is large, i.e. when $L(x)$ and/or $|x|$ is large and/or $|y|$ is small. For the trigonometric functions above the quantities $L(x)$ and $|y|$ were equal to 1 but the argument $x \notin \mathbb{M}$ is large.

A standard approach to improve the performance of computational algorithms is to scale the initial and/or intermediate data to avoid large errors. In computing the matrix exponential $\exp(A)$, $A \in \mathbb{C}(n)$, the scaling $A \to A\,2^{-m}$ is done by a scaling factor 2^{-m}, $m \in \mathbb{Z}_1$, which is a binary degree in order to reduce rounding errors [15].

5.2. Linear Algebraic Equations

An instructive example, illustrating BFCT to the solution of vector algebraic equations $Ax = b$ with integer data, is designed as follows. Let $A \in \mathbb{Z}(n)$ be invertible matrix and $b \in \mathbb{Z}(n, 1)$. The command `A = fix(100*randn(n));` generates a matrix $A \in \mathbb{Z}(n)$ with normally distributed with integer elements of moderate magnitude and zero mean. Choosing the reference solution as `X_0 = ones(n,1)` we get `b = A*X_0`. The computed solution is `X_comp = A\b` with relative error `E_n = norm(X_comp - X_0)/sqrt(n)`.

The a priory relative error estimate is `est = eps*cond(A)`. Usually E_n is a small multiple of est and this may be checked by BFCT. We consider three types of computed solutions as follows.

1. Standard solution `X_1 = A\b` obtained by Gauss elimination with partial pivoting based on LR decomposition $PA = LR$ of A, where L is lower triangular matrix with unit diagonal elements, R is upper triangular matrix with nonzero diagonal elements and P is permutation matrix.
2. Solution `X_2 = R\(Q'*b)` obtained by QR decomposition `[Q,R] = qr(A)` of A, where $A = QR$, the matrix Q is orthogonal and the matrix R is upper triangular.
3. Solution `X_3 = inv(A)*b` based on finding the inverse $\mathrm{inv}(A) = A^{-1}$ of A.

Solution X_1 is computed as a default in MATLAB® and is accurate and fast. Solution X_2 is very accurate although more expensive and is also used. Solution X_3 is not recommended and is included here for tutorial purposes. Indeed, computing `inv(A)` and multiplying the vector b by this matrix are unnecessary operations which are expensive and reduce accuracy.

The above computational techniques are illustrated by the commands

```
n = 1000; A = fix(100*randn(n)); X_0 = ones(n); b = A*X_0;...
X_1 = A\b; [Q,R] = qr(A); X_2 = R\(Q'*b); X_3 = inv(A)*b;...
E_1 = norm(X_0-X_1)/sqrt(n);E_2 = norm(X_0-X_2)/sqrt(n);...
E_3 = norm(X_0-X_3)/sqrt(n);est = eps*cond(A);...
e_1 = E_1/est; e_2 = E_2/est; e_3 = E_3/est; [e_1;e_2;e_3]
```

 doi:10.20944/preprints202412.0780.v1

Each execution of these commands produces different result because of the code `randn`. The computed quantity `e_1` for systems with up to $n = 1000$ unknowns is usually less than 20, the quantity `e_2` is about 40 and the quantity `e_3` is less than 4. More precisely, intensive computations with $n = 1000$ showed that $\text{mean}(e_1) = 18$, $\text{mean}(e_2) = 3$ and $\text{mean}(e_3) = 40$.

Tests with $n = 5000$ had also been done but they require more computational time. For such values of n we have observed average values $\text{mean}(e_1) = 30$, $\text{mean}(e_2) = 3$, $\text{mean}(e_3) = 150$.

The comparison of e_1 and e_2 with e_3 confirms, among others, the tutorial conclusion that solving linear equations by matrix inversion must definitely be avoided. Rather, the opposite is fact. If the inverse matrix $B = A^{-1}$ is needed for some reason, it is found solving n linear equations $Ab_k = i_k$ for the columns b_k of B, where $i_k = I_n(:, k)$ is the kth column if I_n.

Our main observation, based on BFCT, about comparison of the methods of Gauss elimination and QR decomposition for solving linear algebraic equations, is formulated in the next proposition.

Proposition 5. *For matrices of order up to 1000 the QR decomposition method gives error that is between 5 and 10 times less than the error of the Gauss elimination method.*

In addition, the solution of a linear equation $Ax = b$ via QR decomposition is backward stable. Thus Proposition 5 is a message to the developer of MATLAB®, the respected MathWorks, Inc. Corporation.

The Gauss elimination method based on LR decomposition is preferred to the QR decomposition method because it requires twice less floating-point operations. Since fast computational platforms are now widely available, this consideration is no more valid.

As a rule the implementation of the code `x = A\b` gives good results for matrices $A \in \mathbb{R}(n)$ with n of order 10^3. At the same time large errors in x may occur even for $n = 2$.

Example 13. *Consider the algebraic equation $A(m)x = b(m)$, where the matrices $A(m) \in \mathbb{R}(2)$ are defined by the relation (7) below. For $x_{\text{ref}} = [1; 1]$ we have $b(m) = [1 - 10^m; -1 - 10^m]$. The computed results for $m = 2{:}5$ are presented at Table 2. In cases $m = 4, 5$ there is a warning that the matrix $A(m)$ is close to singular or badly scaled since $\text{cond}(A(m))$ is larger than 10^{16}. As shown in Table 2, for these two cases the relative error is indeed 25% and 100%, respectively.*

Table 2. Errors and estimates for low order systems.

m	`err`	`est`	`err/est`
2	3.1601×10^{-9}	9.0612×10^{-8}	0.0349
3	1.6691×10^{-5}	8.9027×10^{-4}	0.0187
4	2.4926×10^{-1}	9.1475	0.0272
5	1.0002	4.6283	0.2161

5.3. Eigenvalues of 2×2 Machine Matrices

The integer 2×2 matrix

$$A = \begin{bmatrix} -100009999 & 100000000 \\ -100020001 & 100010001 \end{bmatrix} \tag{6}$$

has double eigenvalue $\lambda = 1$ and Jordan form $J_2 = I_2 + N_2$. The elements of A are integers of magnitude 10^8. This is far from the the quantity R such that integers $n > R$ may not be machine numbers.

Example 14. *The MATLAB® code `e = eig(A)'` computes the row 2-vector e of eigenvalues of A as $e_{\text{comp}} = [-0.3955, 2.3955]$ instead of $e = [1, 1]$. The relative error is 140% and there is no true digit in the computed vector e_{comp} (even the sign of the first computed eigenvalue is wrong). The trace of A is computed fortunately as*

 doi:10.20944/preprints202412.0780.v1

`trace(A) = 2` *and is the sum of computed eigenvalues. The determinant* $\det(A) = 1$ *is computed wrongly as* `det(A) = 0.2214`.

Surprisingly, the vector p of the coefficients of the characteristic polynomial of A, computed as `p = poly(A)`, is `[1,-2,-0.9475]`. The last element of the computed vector is not the computed determinant 0.2214, which may confuse the user. Moreover, the computed eigenvalues of A are the roots of this wrong characteristic polynomial and hence they are not computed from the Schur form of A as supposed.

Here we recall the modern paradigm in numerical spectral analysis and root finding of polynomials: *the eigenvalues of a matrix are not computed as roots of its characteristic polynomial; rather the roots of any polynomial are computed as the eigenvalues of its companion matrix.*

The Maple [8] code `jordan(A)`, incorporated in MATLAB®, computes correctly the Jordan form $I_2 + N_2$ of A. The vector of eigenvalue condition numbers, computed by the command `condeig(A)`, is $[c, c]$, where $c = 7.1665 \times 10^7$, is relatively large. This indicates that something is wrong. In reality things are even worse since the vector of eigenvalue condition numbers of the matrix A does not exist because A has one linearly independent eigenvector.

At this moment the user may, or may not be aware that something is wrong (with exception of the code `jordan` which is not very popular and is of restricted use). Now comes the time of BFCT to find the spectrum and the Jordan form of A as follows.

The sum $e_1 + e_2$ of the eigenvalues of A is equal to the trace $A(1,1) + A(2,2) = 2$ which is computed exactly. If the eigenvalues are close or equal, because the eigenvalue condition numbers are large, then we should have $e_1 + e_2 \simeq 2e_1 = 2$ and $e_1 = e_2 = 1$. The Jordan form of A is then either J_2 or I_2. But the only matrix with Jordan form I_2 is I_2 itself and hence the Jordan form of A must be J_2. We have obtained the correct result using BFCT and simple logical reasoning.

The matrix (6) may be written as

$$A(m) = \begin{bmatrix} 1 - a - a^2 & a^2 \\ -1 - 2a - a^2 & 1 + a + a^2 \end{bmatrix}, \; a = 10^m \tag{7}$$

for $m = 4$. Thus we have a family of matrices $\{A(m)\}$ parametrized by the integer m. The trace of $A(m)$ is equal to 2, the determinant of $A(m)$ is equal to 1, the eigenvalues of $A(m)$ are $e_1 = e_2 = 1$ and the Jordan form of $A(m)$ is $I_2 + N_2$.

Below we give the values of the coefficients $c_2 = \mathrm{tr}(A) = 2$ and $c_3 = \det(A) = 1$ of the characteristic polynomial $\lambda^2 - c_2\lambda + c_3$ of A as found by the code `poly(A)`, and of the eigenvalues $e_1 = e_2 = 1$ of A, as computed by the code `eig(A)` in MATLAB®.

1. For $m = 2$ we have $c_2 = 2.0000$, $c_3 = 1.0000$ and $e_{1,2} = 1.0000 \pm 0.0001\,\mathrm{i}$, where the quantity 1.0000 has at least 5 true decimal digits and $0.0001\,\mathrm{i}$ is a small imaginary tail which may be neglected. This result seems acceptable.
2. For $m = 3$ we have $c_2 = 2.0000$, $c_3 = 1.0000$ and $e_1 = 0.9937$, $e_2 = 1.0063$. With certain optimism these results may also be declared as acceptable.
3. For $m = 4$ a computational catastrophe occurs with $c_2 = 2.0000$ (true), $c_3 = -0.9475$ (completely wrong) and $e_1 = -0.3955$, $e_2 = 2.3955$ (also completely wrong). Here surprisingly $\det(A)$ is computed as 0.2214 which differs from 1 (which is to be expected) but differs also from c_3. It is not clear what and why had happened. Using the computed coefficients c_2, c_3, the roots of the characteristic equation $\lambda^2 - c_2\lambda + c_3 = 0$ now are $\lambda_1 = 0.7709$, $\lambda_2 = 1.2291$ instead of the computed e_1 and e_2. This is a strange wrong result.
4. We give also the results for $m = 5$ which are full trash and are served without any warning for the user. We have $c_2 = 2.0000$, $c_3 = 5591.5$ and $e_{1,2} = 1.0000 \pm 74.770\,\mathrm{i}$. We also get $\det(A) = -4472.2$ for completeness.

 doi:10.20944/preprints202412.0780.v1

In all cases the sum $c_2 = 2.0000$ of the wrongly computed eigenvalues is almost exact which is a well known fact in the numerical spectral analysis of matrices [15].

The conclusion from the above considerations is that BFCT in this case, in contrast to the standard software, always give $e_1 \simeq e_2 = c_2/2 = 1.0000$ with 5 true decimal digits. This is a good approximation to the eigenvalues $e_1 = e_2 = 1$ of the matrices $A(m)$ for $m = 3, 4, 5$ when the codes `eig` and `poly` fail.

6. Zeros of Functions

Computing zeros of functions and of polynomials in particular is a classical problem in numerical analysis. Although powerful sophisticated computational algorithms for this purpose have been developed, the use of BFCT in this area is still useful.

Consider first the scalar case. Computing zeros of functions $f(x) = 0$, where $f : \mathbb{R} \to \mathbb{R}$, in a given interval $T = [a, b]$, where f is a continuous function, may be a difficult problem. Here we distinguish three main tasks.

1. Find one particular solution r_1 of equation $f(x) = 0$ in the interval T.
2. Find the general solution $f^{-1}(0) \cap T$ of equation $f(x) = 0$ in the interval T.
3. Find all roots $[r_1, r_2, \ldots, r_n]$ of a given nth degree polynomial $P(x)$.

The condition $f(a)f(b) < 0$ guarantees that there is at least one solution in the interval (a, b). Whether f has zeros at the points a, b is checked by direct computation.

Solving real and complex polynomial equations $P(x) = 0$ of nth of the form

$$P(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1} = 0$$

and $c_1 \neq 0$ is done by the code `vpasolve(P(x) == 0)` from MATLAB®. It finds all roots of $P(x)$ with quadrupole precision of 32 true decimal digits.

For $n = 100$ the speed of the corresponding software on a medium computer platform is still acceptable. For $n = 1000$ and higher the performance of the code may be slow even on fast platforms and hence it may not be applicable to RTA (if such computations are ever needed in RTA). Note that we write this in end of year 2024.

A faster solution may be obtained by the code `roots(p)`, where p is the vector of coefficients $p_1, p_2, \ldots, p_{n+1}$ of $P(x)$. This code computes the column vector of roots $r = [r_1; r_2; \ldots; r_n] \in \mathbb{C}(n, 1)$ of $P(x)$ as the vector of eigenvalues of the companion matrix $\mathrm{C}(A)$ of $P(x)$ using the QR algorithm, i.e. `r = eig(compan(p))`. The code `roots(p)` works relatively fast for n up to several thousands but may produce wrong results even for small values of n when the polynomial has multiple roots and the computational problem is poorly conditioned.

It is known that relative errors in computing multiple roots of equations $f(x) = 0$ in FPA, where f has a sufficient number of derivatives around a root $x = r$, are of order $\mathrm{eps}^{1/\mathrm{n}}$, where n is the multiplicity of r. Correspondingly, relative errors for most computational algorithms for solving such equations are of order $\gamma\,\mathrm{eps}^{1/\mathrm{n}}$, where γ is a constant of order $\mathrm{O}(1)$.

To estimate γ, consider the equation $P(x) = 0$ when the nth degree polynomial $P(x)$ has n-tuple root $r_1 = 1$. Denote the relative error in the computed vector r_{comp} as

$$E_n = \frac{\|r_{\mathrm{comp}} - \mathrm{ones}(n, 1)\|}{\sqrt{n}}$$

The ratio $\gamma_n = E_n/(\mathrm{eps})^{1/\mathrm{n}}$ is shown at Table 3 for the extended form $x^n - nx^{n-1} + \cdots + (-1)^n$ of the polynomial $P(x) = (x-1)^n$, and values of n from 3 to 20. For $n = 2$ the computed roots are exact and $\gamma_2 = 0$.

 doi:10.20944/preprints202412.0780.v1

Table 3. Constants in error estimates for code `roots`.

n	3	4	5	6	7	8	9	10	11
γ_n	1.73	1.78	1.53	1.95	1.62	1.60	1.84	1.72	2.03
n	12	13	14	15	16	17	18	19	20
γ_n	1.76	2.04	1.94	2.10	1.92	2.06	2.01	1.97	1.91

We see that the relative error E_n in the computed solution for n up to 20 satisfies the heuristic estimate $E_n \le \gamma_n \mathrm{eps}^{1/n}$, where the average of γ_n is 1.87. Next we assume that $\gamma_n = 1.9$ for simplicity.

The polynomial $P(x) = (x-1)^n$ may be found using the command `expand((x-1)^n)`, or asking the intelligent dialogue computer system WolframAlpha [10]

```
Find the coefficients of the polynomial (x-1)^n
```

We stress that in all cases the code `vpasolve(P(x) == 0)` gives the exact solution vector `ones(n,1)`.

Usually the computed roots are unacceptable if their relative error is greater than 1%. For example, it follows from Table 3 that the root $r = 1$ of multiplicity 20 is computed by the code `roots` with relative error 32% which is unacceptable. It must be stressed that this is not due to program disadvantages of the code but rather to high sensitivity of the eigenvalues of the companion matrix $\mathrm{C}(A)$ to perturbations in A.

If the polynomial equation $P(x) = 0$ has a root of multiplicity n the relative error is estimated as $1.9\,\mathrm{eps}^{1/n}$. Solving the equation $1.9\,\mathrm{eps}^{1/n} = 0.01$ we get $n = 6.89$ and $n \ge 7$. We summarize this slightly unexpected observation in a special proposition.

Proposition 6. *Relative errors in computing n-tuple roots of polynomials by the code* `roots` *may be unacceptable when* $n \ge 7$.

Let now a continuous function f be given and suppose that we look for all roots of the equation $f(x) = 0$ in the interval $T = [a, b]$, or for the set $f^{-1}(0) \cap T$. Here use of BFCT seems necessary. A direct approach is to plot the graph of the function $x \mapsto |f(x)|$, $x \in T$. Let N be a sufficiently large integer, say $N = 1000$. We may use the commands `X = linspace(a,b,N); Y = abs(f(X));` and `plot(X,Y)` to plot the graph of the function $x \mapsto |f(x)|$, $x \in T$.

The zeros of f are marked as inverted peaks at the roots $r_1 < r_2 < \cdots$. Next the codes `vpasolve(f,r_k)` or `fzero(f,r_k)` may be used around each of the points r_k to specify the roots with full accuracy. This approach is well known in the computational practice and is recommended in some MATLAB® guides.

More difficult problems are the solutions of nonlinear vector equations $f(x) = 0$ and of matrix equations $F(X) = 0$, where x and $f(x)$ are n-vectors, and X and $F(X)$ are $n \times n$ matrices. The MATLAB® code `fsolve` is intended to solve nonlinear vector problems, while the codes `are`, `axxbc`, `care`, `dare`, `dlyap`, `dric`, `lyap`, `ric`, `sylv` and others solve linear and nonlinear matrix equations. The performance of all these codes may be checked by BFCT and CPRS. This a challenging problem which will be considered elsewhere.

7. Minimization of Functions

7.1. Functions of Scalar Argument

The minimization of the expression $y = f(x)$, $x \in T = [a, b]$, where $f : T \to \mathbb{R}$ is a continuous function, is done in MATLAB® by a variant of the method of golden ratio. The corresponding code `[x_0,y_0] = fminbnd(f,a,b)` finds a local minimum in the interval T, where $y_0 = f(x_0)$.

To find the global minimum $[x_{\min}, y_{\min}]$ of f in the interval T we may use the graph of the function f plotted by the commands `x = linspace(a,b,n); y = f(x); plot(x,y).` Then the approximate global minimum $[c, f(c)]$ of f is determined visually, considering also the pairs $[a, f(a)]$ and $[b, f(b)]$ at the end points of T. The code

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 December 2024 doi:10.20944/preprints202412.0780.v1


```
[x_min,y_min] = fminbnd(f,c-h,c+h)
```

is then used in a neighborhood of the approximate minimum $[c, f(c)]$. This approach requires human interference in the process and may be avoided as shown later.

The problem with the code `fminbnd` is not that it may find local minimums. The real problem with this and similar codes is that they can miss the global minimum even when the function f is unimodal, i.e. when it has a unique minimum. So BFCT for such functions is a must. The problem is that usually we do not know whether the function is unimodal or not and whether the computed minimum is minimum at all. The more so when performing complicated algorithms in RTA.

Example 15. *Consider the function*

$$f(x) = 2 - x^2 \exp(1 - x^2),\ x \in [0, \infty)$$

This function is unimodal with minimum $[x_{\min}; y_{\min}] = [1; 1]$. *In some cases this minimum is found as*

```
[x_min,y_min] = fminbnd(f,0,b)
```

for some $b > 0$. *But in other cases it is not found correctly and here we should use BFCT. The code works well for* $1 \le b \le 16.85$ *but gives a completely wrong result for* $b \ge 16.86$ *without warning. Indeed, we have the good result*

```
[x_min;y_min] = fminbnd(f,0,16.85) =  0.999992123678799
                                      1.000000000124073
```

and the bad result

```
[x_min;y_min] = fminbnd(f,0,16.86) = 16.859937887512295
                                      2
```

The second result has 1586% relative error for $x_{\min}$ *and 100% relative error for* $y_{\min}$.

The reason for the disturbing events in Example 15 is complicated. We may expect that $f(x) = 2$ for values of x such that the expression $\varphi(x) = x^2 \exp(1 - x^2)$ is less than ρ since then $f(x) = 2 - \varphi(x)$ is rounded to 2. The root $r_0 > 0$ of the equation $\varphi(x) = \rho$ is $r_0 = 6.38$ and we indeed have $\varphi(r_0) = 2$. But r_0 is much less then 16.86 and hence the code `fminbnd` works with certain sophisticated version of VPA instead with FPA. In particular, the quantity $\alpha = \varphi(16.86)$ is much less than ρ with $\alpha/\rho \simeq 2.46 \times 10^{-105} \ll 1$. We shall summarize the above considerations in the next statement.

Proposition 7. *The software for minimization of functions of one variable such as* `fminbnd(f,a,b)` *from MATLAB® may fail to produce correct results even for some unimodular functions* $f : T \to \mathbb{R}$.

In contrast, BFCT can always be used to find the global minimum automatically. Let the function $f : T \to \mathbb{R}$ be unimodal. Consider the vectors `X = linspace(a,b,n+1)` with elements $X(k) = a + (k-1)(b-a)/n$ and $Y = f(X)$ with elements $Y(k)$. The MATLAB® command `[y,m] = min(Y)` computes an approximation $[X(m), y]$ to the minimum $[x_{\min}, y_{\min}]$. Finally, the (almost) exact minimum is computed between the neighboring elements of $X(m)$ as

```
[x_min,y_min] = fminbnd(f,X(m-1),X(m+1))
```

7.2. Functions of Vector Argument

For $n \ge 2$ the minimization of continuous functions $f : \mathbb{R}(n, 1) \to \mathbb{R}$ of vector argument meets difficulties similar to those described in Section 7.1. And other specific difficulties as well. Unconstraint minimization $f(x) \to \min$ is done in MATLAB® by the Nelder-Mead algorithm [23,24], realized by

 doi:10.20944/preprints202412.0780.v1

the code `fminsearch` [6]. We stress that the code finds a local minimizer $[x_{\min}, y_{\min}]$ of the function f, where $y_{\min} = f(x_{\min})$.

The function f may not be differentiable which is an advantage of this algorithm. The algorithm compares the function values $f(x)$ at the vertexes of a $(n+1)$-simplex and then replaces the vertex with highest value of f by another vertex. The simplex usually (but not always, as shown below) contracts on a minimum of f which may be local or global. The corresponding command is

```
[x_min,y_min] = fminsearch(f,x_0)
```

where x_0 is the initial vector.

The computed result depends on the next five factors, among others:

1. the computational problem
2. the computational algorithm
3. the FPA
4. the computer platform
5. the starting point x_0

Factors 1-3 are usually out of control of the user. Factor 4 is partially under control, e.g. the user may buy another platform. Only factor 5, which is very important, is entirely under control. As mentioned above, the computed value $x_{\min}$ depends on the starting point x_0. This is denoted as $x_{\min} = \Psi(x_0)$.

Definition 10. *The point x_0 is said to be a numerical fixed point (NFP) of the computational procedure if $x_0 = \Psi(x_0)$.*

The computed result $x_{\min}$ is NFP of the computational procedure. NFP depends not only on the computational problem and the computational algorithm but also on the FPA and on the particular computer platform, where the algorithm is realized. Unfortunately, computed values for $x_{\min}$ which are far from any actual minimum are often NFP of the computational procedures. They are served without warning to the user.

For a class of optimization problems the function f has the form

$$f(x) = c + p(x)e(x),\ e(x) = \exp\left(-a_1x_1^2 - a_2x_2^2 - \cdots - a_nx_n^2\right) \tag{8}$$

where $c > 0$, $p(x)$ is a polynomial in x and a_k are positive constants. Since $f(x) \to c$ when $\|x\| \to \infty$ the function (8) has (at least one) global minimum $[x_{\min}, y_{\min}]$.

Numerical experiments with functions f of type (8) show that not only the computed valus for $x_{\min}$ and $y_{\min}$ depend on the initial guess x_0 but that in many cases the code `fminsearch` produces a wrong solution for $x_{\min}$ which is far from any minimizer of the function f. The reason is that the value of $p(x)e(x)$ may become very small relative to c. Then $f(x)$ is computed as c and the algorithm stops at a point which is far from any minimizer of f. Usually, no warning is issued for the user in such cases.This may be a problem when the minimization code is a part of an automatic computational procedure which is not under human supervision. In such cases application of the techniques of AI and MAI may be useful.

Example 16. *Let $n = 2$ and consider the function of type (8)*

$$f(x) = 2 - x_1x_2\exp\left(1 - \frac{1}{2}(x_1^2 + x_2^2)\right)$$

i.e. $c = 2$, $p(x) = -\exp(1)x_1x_2$ and $a_1 = a_2 = 1/2$. The function f is coded as

```
f = @(x)2-x(1)*x(2)*exp(1-x(1)^2/2-x(2)^2/2)
```

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 December 2024 doi:10.20944/preprints202412.0780.v1

The function $y = f(x)$, $x \in \mathbb{R} \times \mathbb{R}$, *has two global minimizers*

$$x_{\min,1} = [1,1],\ x_{\min,2} = [-1,-1]$$

for which

$$y_{\min,1} = y_{\min,2} = 1.$$

It also has two global maximizers

$$x_{\max,1} = [1,-1],\ x_{\max,2} = [-1,1]$$

for which

$$y_{\max,1} = y_{\max,2} = 3.$$

Let the initial guess for the code

```
[x_min,y_min] = fminsearch(f,x_0)
```

be $x_0 = [c,c]$. *For* $c = 6.4308$ *we get the relatively good result*

```
x_min = -1.000041440968428    -0.999998218805417
y_min =  1.000000001720503
```

For the slightly different value $c = 6.4309$ *the computed result* $x_{\min} = [c,c]$ *is a numerical fixed point which is not a minimizer. Indeed, we have*

```
x_min = 6.430900000000000    6.430900000000000
y_min = 2.000000000000000
```

The second computed result in Example 16 is a numerical catastrophe with 543% relative error in $x_{\min}$ and 100% relative error in $y_{\min}$. The reason is that $c^2 \exp(1-c^2) \le \rho$ for $c \ge 6.4391$ and $f(c)$ is rounded to 2. The conclusion is that the algorithm of the moving simplex uses FPA rather than VPA; compare with Example 15.

The minimization problem in Example 16 is not unimodal since it has two solutions for $x_{\min}$ and a unique solution for $y_{\min}$. Solving unimodal problems by the code `fminsearch` also leads to large errors. Consider the next unimodal problem for which the code also fails for moderate data.

Example 17. *Let* $n = 2$ *and consider the function of type (8)*

$$f(x) = 2 - x_1 \exp\left(\frac{1}{2} - \frac{1}{2}x_1^2 - (x_2-1)^2)\right)$$

The function f *is coded as*

```
f = @(x)2-x(1)*exp(1/2-x(1)^2/2-(x(2)-1)^2)
```

The minimization problem has unique solution $x_{\min} = [1,1]$, $y_{\min} = 1$. *For* $c = 5.81$ *the code*

```
[x_min,y_min] = fminsearch(f,[c,c])|
```

gives a quite acceptable result, namely

```
x_min =  1.000039724696251    1.000017661431510
y_min =  1.000000001889957
```

For a slightly different value of c the computed result $x_{\min} = [c,c]$ *is a numerical fixed point which is not a minimizer. Indeed, for* $c = 5.82$ *the computed result is wrong, namely*

 doi:10.20944/preprints202412.0780.v1


```
x_min =  5.820000000000000   5.820000000000000
y_min =  2
```

There is 482% relative error in the computed argument $x_{\min}$ and 100% relative error in the computed minimum $y_{\min}$.

The output of such situations is, at least for small n, to use BFCT as follows. Let $n = 2$ and let $f : T = T_1 \times T_2 \to \mathbb{R}$ be the function which has to be minimized, where $T_k = [a_k, b_k] \subset \mathbb{R} \times \mathbb{R}$. We choose positive integers n_1, n_2 and compute the grids

```
X_k = linspace(a_k,b_k,n_k)
```

and the quantities

$$A(i,j) = f(X_1(i), X_2(j)).$$

Thus we construct the matrix $F \in \mathbb{R}(n_1, n_2)$ which is a discrete analogue of the surface $z = f(x,y)$, $(x,y) \in T$.

Next the minimal element $F(k_1, k_2)$ and its indexes (k_1, k_2) are found by the BFCT described in Section 4.2. Now the approximate global minimum of f is $[X_{\min}, Y_{\min}]$, where $X_{\min} \in \mathbb{R}(2,1)$ may be used as the starting point x_0 for the code

```
[x_min,y_min] = fminsearch(f,x_0)
```

Extensive numerical experiments show that this code now works properly.

8. Canonical Forms of 2×2 Matrices

8.1. Preliminaries

An interesting observation is that important properties of linear operators in n-dimensional vector spaces can be revealed for dimensions as low as $n = 2$, see [28]. In this section we consider some modified definitions of canonical forms and present instructive examples for the sensitivity of Jordan forms using BFCT.

8.2. Jordan and Generalized Jordan Forms

In this section we use some not very popular definitions of canonical forms, e.g. a definition of Jordan form of a matrix without mentioning eigenvalues and without using the concept of Jordan blocks.

Definition 11. *Let $n \in \mathbb{Z}_2$. The upper triangular bi-diagonal matrix $J \in \mathbb{C}(n)$ is said to be a Jordan matrix if*

1. $J(k,k+1) \in \{0,1\}$
2. $J(k,k+1) = 1$ *implies* $J(k,k) = J(k+1,k+1)$
3. $J(k,k) \neq J(k+1,k+1)$ *implies* $J(k,k+1) = 0$

for $k = 1,2,\ldots,n-1$. The set of Jordan matrices is denoted as $\mathbb{J}(n)$.

Definition 12. *The Jordan matrix J is spectrally ordered if $J(k,k) \preceq J(k+1,k+1)$ for $k = 1,2,\ldots,n$. The set of spectrally ordered Jordan matrices is denoted as $\widetilde{\mathbb{J}}(n)$.*

Proposition 8. *The set $\mathbb{J}(2)$ consists of matrices* $\mathrm{diag}(\lambda,\mu)$ *and $\lambda I_2 + E_{1,2}$, where $\lambda, \mu \in \mathbb{C}$.*

Definition 13. *The Jordan problem for a nonzero matrix $A \in \mathbb{C}(n)$ is to find a pair $(X_A, J_A) \in GL(n) \times \mathbb{J}(n)$ such that $X_A J_A = A X_A$. Here X_A is the transformation matrix and J_A is the Jordan form of A.*

 doi:10.20944/preprints202412.0780.v1

Note that we do not use explicitly the spectrum of A or the concept of Jordan blocks. Indeed, given a general matrix A (even with integer elements and of low size as in (6)) then a priori we know neither its spectrum nor its Jordan structure.

Remark. *The Jordan form J_A is not unique and is defined only by the order of units on its bi-diagonal unless $A = \lambda I_n$ when $J_A = A$ is unique and X_A is any invertible matrix.*

The Jordan form may be defined uniquely to become a canonical Jordan form in terms of Group theory [29] and Integer partition theory [30] although this is rarely done in sufficient extent in the literature. The transformation matrix X_A is always non-unique since e.g. the matrix γX_A, $\gamma \neq 0$, is also a transformation matrix.

Example 18. *Zz If the matrix $A \in \mathbb{C}(n)$ has a single eigenvalue then there are 2^{n-1} different orderings of the unit elements on the $n-1$ positions $(k, k+1)$ of the super diagonal of A.*

Definition 14. *Let $J \in \mathbb{J}(n)$. The matrix $X \in GL(n)$ is said to be a stabilizer of J if $X^{-1}JX \in \mathbb{J}(n)$. The set of stabilizers of A is denoted as $\mathrm{Stb}(A) \subset GL(n)$. The matrix I_n is the center of $\mathrm{Stb}(A)$.*

Note that the set $\mathrm{Stb}(A)$ in general is not necessarily a group unless $A = \lambda I_n$ and $\{I_n\}$ is not a center in group-theoretical sense (we recall that the center of $GL(n)$ is the subgroup of matrices $a\mathcal{I}_n$, $a \neq 0$).

Example 19. *Let $n = 2$ and $A \in \mathbb{J}(2)$. Then the following three cases are possible.*

1. *If $A = \lambda I_2$ then $\mathrm{Stb}(A)$ is the group $GL(2)$.*
2. *If $A = \mathrm{diag}(\lambda_1, \lambda_2)$, $\lambda_1 \neq \lambda_2$, then $\mathrm{Stb}(A)$ is the set of matrices $\mathrm{diag}(a_1, a_2)$ and $a_1 E_{1,2} + a_2 E_{2,1}$, where $a_1 a_2 \neq 0$.*
3. *If $A = \lambda I_2 + E_{1,2}$ then $\mathrm{Stb}(A)$ is the set of matrices $a(E_{1,2} + E_{2,1})$, $a \neq 0$.*

Example 20.

Definition 15. *The upper triangular bi-diagonal matrix $G \in \mathbb{C}(n)$ is said to be a generalized Jordan matrix if*

1. *$G(k, k+1) \neq 0$ implies $G(k,k) = G(k+1, k+1)$*
2. *$G(k,k) \neq G(k+1, k+1)$ implies $G(k, k+1) = 0$.*

The set of generalized Jordan matrices is denoted as $\mathbb{G}(n)$.

Generalized Jordan matrices G have the structure of Jordan matrices, where the nonzero elements $G(i, i+1)$ (if any) are not necessarily equal to 1. Hence they are less sensitive to perturbations in the matrix A.

There are two forms $\mathrm{diag}(\lambda, \mu)$ and $\lambda I_2 + aE_{1,2}$ of the elements of $\mathbb{G}(2)$, where $\lambda, \mu, a \in \mathbb{C}$ and $a \neq 0$. Note that we do not define spectrally ordered generalized Jordan matrices since they are not important for the computational practice.

Both Jordan matrices J and generalized Jordan matrices G are bi-diagonal, where the bi-diagonal elements of J are zeros or ones, while the bi-diagonal elements of G are correspondingly zeros or nonzero numbers.

Definition 16. *The generalized Jordan problem for a nonzero matrix $A \in \mathbb{C}(n)$ is to find a pair $(Y_A, G_A) \in GL(n) \times \mathbb{G}(n)$ such that $Y_A G_A = A Y_A$. Here Y_A is the transformation matrix and G_A is the generalized Jordan form of A.*

Consider now perturbations in Jordan forms. Let $A(\varepsilon) = A_0 + \varepsilon A_1$, where $A_0, A_1 \in \mathbb{C}(n)$ and $\varepsilon \in (-\varepsilon_0, \varepsilon_0)$ with $\varepsilon_0 > 0$ being a small parameter. If the matrix A_0 has multiple eigenvalues then the

 doi:10.20944/preprints202412.0780.v1

transformation matrix $X(\varepsilon) = X_{A(\varepsilon)}$ and the Jordan form $J(\varepsilon) = J_{A(\varepsilon)}$ of $A(\varepsilon)$ may be discontinuous at the point $\varepsilon = 0$. The situation with the transformation matrix $X(\varepsilon) = X_{A(\varepsilon}$ is even more subtle.

If the matrix A_0 has simple eigenvalues then the matrices $X(\varepsilon)$ and $J(\varepsilon)$ depend continuously on ε. In particular $X(0) = X_0$ and $J(0) = J_0$.

Next we consider matrices $A \in \mathbb{R}(2)$ with real spectrum. Then the transformation matrices X, Y, the Jordan forms J and the generalized Jordan forms G are real as well.

For small size matrices A with rational elements (machine elements in particular) $A(i,j)$ the matrices X_A and J_A are computed by the MATLAB® (Maple) command `[X_A,J_A] = jordan(A)` exactly.

Example 21. *Consider the matrix $A = I_2$ and let*

$$A(\varepsilon) = I_2 + \varepsilon E_{2,1},\ \varepsilon = 2^{-m},\ m \in \mathbb{Z}_1 \neq 0.$$

Then the transformation matrix and the Jordan form of $A(\varepsilon)$ are

$$X(\varepsilon) = E_{2,1} + \varepsilon^{-1} E_{1,2},\ J(\varepsilon) = I_2 + E_{1,2}.$$

The code `[X,J] = jordan(A)` *computes the matrices $X(\varepsilon)$, $J(\varepsilon)$ exactly for $m \leq 1074$. For $m = 1075$ the matrices X_A and J_A are computed as I_2 since ε is rounded to zero.*

Example 22. *Consider the matrix $A = I_2 + E_{1,2}$ and let*

$$A(\varepsilon) = I_2 + \varepsilon E_{1,1},\ \varepsilon = 2^{-m},\ m \in \mathbb{Z}_1 \neq 0.$$

Then the Jordan forms of A are

$$J_1(e) = \mathrm{diag}(1+\varepsilon, 1),\ J_2(\varepsilon) = \mathrm{diag}(1, 1+\varepsilon).$$

The form $J_1(\varepsilon)$ is unachievable. For the form $J_2(\varepsilon)$ we have infinitely many transformation matrices $X_2(\varepsilon)$, e.g.

$$X_2(\varepsilon) = \mu E_{1,2} - E_{2,1} - \varepsilon^{-1} E_{1,1}$$

where $\mu \neq 0$ is arbitrary.

The code `[X,J] = jordan(A)` *computes exactly the matrices $X_2(\varepsilon)$ (with $\mu = 1$) and $J_2(\varepsilon)$ for $\varepsilon = 2^{-m}$ and $m \leq 44$. For $m = 45$ these matrices are computed wrongly as $X_2(\varepsilon) = I_2$ and $J_2(\varepsilon) = I_2$.*

Proposition 9. *Jordan forms J_A of matrices A with multiple eigenvalues are sensitive to perturbations in A for three main reasons: 1) because $J_A(i,j) = 0$ for $i \geq j+1$; 2) because $J(i,j) = 0$ for $j \geq i+2$ (if $n \geq 3$) and 3) because their nonzero elements $J_A(i, i+1)$ (if any) are equal to 1.*

8.3. Condensed Schur Form

Schur forms S_A of matrices A satisfy the only condition $S_A(i,j) = 0$ for $i \geq j+1$ and are this less sensitive to perturbations in the nevertheless they may be discontinuous in a neighborhood of a matrix A_0 with multiple eigenvalues. The case of simple eigenvalues is studied in more detail, see e.g. [31].

Definition 17. *The matrix $S \in \mathbb{C}(n)$ is said to be a Schur matrix if it is upper triangular. The set of Schur matrices is denoted as $\mathbb{S}(n)$.*

We recall that we study only cases $n \geq 2$. Now it is easy to see that $\mathbb{S}(n) = \mathbb{G}(n)$ if and only if $n = 2$.

 doi:10.20944/preprints202412.0780.v1

Each matrix $A \in \mathbb{C}$ is unitary similar to a Schur matrix $S_A \in \mathbb{S}(n)$ such that $U_A S_A = A U_A^{\mathrm{H}}$ for some matrix $U_A \in U(n)$. The matrix U_A is the transformation matrix and the matrix $S_A \in U(n)$ is the condensed Schur form of A. The Schur problem for A is to describe the set of pairs (U_A, S_A) and to study their sensitivity relative to perturbations in A.

Although less sensitive to perturbations in A, the solution (U_A, S_A) of the Schur problem may also be sensitive and even discontinuous as a function of A at points A_0, where the matrix A_0 has multiple eigenvalues.

Let $A = \lambda I_2$ and εE_A be a perturbation in A. The Schur form of A is A itself and in this case we may consider the trivial solution (I_2, A) of the Schur problem for A. For arbitrary small $\varepsilon \neq 0$ the Schur forms of $A + \varepsilon E_{2,1}$ are $S = I_2 \pm \varepsilon E_{1,2}$ with transformation matrices $V = \pm E_{1,2} \pm E_{2,1}$. Thus the solution changed from the trivial solution $U = I_2$ for V for $\varepsilon \neq 0$.

Example 23. *Let $m = 50$, $a = 2^{-m}$ and $A = [1, 0; a, 1]$. The command*

```
[X,J] = schur(A)
```

gives the correct answer

```
X =  0    -1        J = 1.000000000000000  -0.000000000000001
     1     0            0                   1.000000000000000
```

For $m = 51$ we have

```
X =  1     0        J = 1     0
     0     1            0     1
```

due to rounding errors. Obviously the code `schur` *works with FPA.*

9. Least Squares Revisited

9.1. Preliminaries

In least squares methods (LSM) the aim is to minimize the sum of squared residuals. A not very popular fact is that a smaller residual may correspond to a larger error [21,22]. Moreover, this phenomenon may be observed on an infinite set of residual and errors. This may have far going consequences. So the LSM has to be revisited taking into account such cases.

A least squares problem (LSP) may arise naturally in connection with a given approximation scheme, or as an equivalent formulation of a zero finding method.

9.2. Nonlinear Problems

The solution of the (overdetermined) equation $f(x) = 0$, where x and $f(x)$ are vectors and f is a continuous function, may be reduced to minimization of the quantity $\|f(x)\|^2$. Such minimization problems arise also in a genuinely optimization statement in e.g. approximation of data.

Let $\varphi \colon \mathbb{C}(n,1) \to \mathbb{R}_+$ be a continuous function and let the problem $\varphi(x) \to \min$ be unimodal, i.e. there is a unique (global) minimum $[x_0; y_0] \in \mathbb{C}(n,1) \times \mathbb{R}_+$, where $y_0 = \varphi(x_0)$. Denote by $e_k = \|x_k - x_0\|$ the distance between the vector x_k and x_0, or the absolute error of the vector x_k when x_k is interpreted as a computed solution.

Let $n = 1$ and $\varphi \colon \mathbb{R} \to \mathbb{R}_+$. Let (x_k) be a sequence of computed solutions. Then it is possible that $r_k = \varphi(x_k)$ tends to 0 and e_k tends to ∞ when $k \to \infty$, as the next example shows.

Example 24. *Let*

$$\varphi(x) = 1 + \frac{(x-1)^2}{1 + (x-1)^4}, \; x \in \mathbb{R}$$

The function φ is unimodal with minimum $[x_0, y_0] = [1, 1]$. Let $x_k = k$. Then $r_k = \varphi(x_k) \to 0$ and $e_k = |k - 1| \to \infty$ as $k \to \infty$.

 doi:10.20944/preprints202412.0780.v1

This result is easily extended to the case $n \geq 2$.

Example 25. *Let*

$$\varphi(x) = 1 + \frac{(\|x\| - 1)^2}{1 + (\|x\| - 1)^4},\ x \in \mathbb{C}(n, 1)$$

The function φ has infinitely many minimums $[x_0, 1]$ on the unit sphere $\|x_0\| = 1$. Let $x_k = k[1; 0; \ldots; 0]$ for $k \in \mathbb{Z}_1$, i.e. $\|x_k\| = k$. Then $r_k \to 0$ and $e_k \to \infty$ as $k \to \infty$.

Thus the residual r_k and the error e_k may have opposite behavior. This phenomenon, namely larger the remainder r_k, smaller the error e_k, is known as *Remainders vs. Errors*, see [22]. It may concern the stopping rule in the implementation of LSM.

9.3. Linear Problems

Consider the matrix $A \in \mathbb{R}(m, n)$ of full column rank $m > n$ and let $b \in \mathbb{R}(m, 1)$ be a given vector which is not in the range of A. The linear least squares problem (LLSP) is to find the vector $x \in \mathbb{R}(n, 1)$ such that $r(x) = \|Ax - b\| \to \min$. This problem is unimodal and its theoretical solution is $A^\dagger b$, where $A^\dagger = (A^\top A)^{-1} A^\top \in \mathbb{R}(n)$ is the pseudo-inverse of the matrix A.

This representation of the solution is not used in computational practice since it is ineffective and may be connected with substantial loss of accuracy. Instead, the solution is found by QR decomposition $A = QR$ of the matrix A. Let $R = [S; T]$ and $Q^\top b = [c; d]$, where the matrix $S \in \mathbb{R}(n)$ is upper triangular and invertible, and $c \in \mathbb{R}(n, 1)$. The solution x^* is obtained by the code `x = A\b`, which in fact computes `x = S\c`.

Intensive experiments based on BFCT with random data show that the error of the "bad" solution $A^\dagger b$ is between 5 and 10 times larger than the error of the " good" solution $x^* = A\backslash b$. In addition, finding x_0 requires much more computational time.

Let $x_1, x_2 \in \mathbb{R}(n, 1)$ be two approximate solutions of the above LLSP which are different from the exact solution x_0. Set $r_k = \|Ax_k - b\|$ and $e_k = \|x_k - x_0\|$, $k = 1, 2$. We have $\sigma_n \leq r_k / e_k \leq \sigma_1$, where $\sigma_1 = \|A\|$ and $\sigma_n = 1/\|A^{-1}\|^{-1}$ are the maximal and minimal singular values of A. Next, it is possible [22] to choose x_k so as $r_1 / e_1 = \sigma_1$ and $r_2 / s_2 = \sigma_n$. It is fulfilled

$$\frac{r_1}{r_2} = \mathrm{cond}(A) \frac{e_1}{e_2},\ \mathrm{cond}(A) = \frac{\sigma_1}{\sigma_n} \tag{9}$$

We see that when the matrix A is poorly conditioned in the sense that $\mathrm{cond}(A) \gg 1$ the behavior of remainders and errors is opposite: larger the remainder, smaller the error. This fact deserves a separate formulation.

Proposition 10. *When $\mathrm{cond}(A)$ is large, larger remainders may correspond to smaller errors. In this case checking the accuracy of the computed solutions by the remainders is misleading.*

The situation with remainders and errors is indeed ironic. If the matrix A is well conditioned with $\mathrm{cond}(A)$ close to 1 the computed approximations are most probably good and there is no need to check their accuracy by remainders. But if the matrix A is poorly conditioned and the accuracy test seems necessary, the accuracy check by remainders may lead to wrong conclusions. Note that this phenomenon is possible only if $n \geq 2$.

Example 26. *Consider the simplest case $n = 2$ and $A = [B; 0] \in \mathbb{R}(m, 2)$, $b = 0$, where $B = [\mu^3, -1; \mu^3, 1] \in \mathbb{R}(2)$ and $\mu > 0$ is a small parameter. The only solution of the LSP is $x_0 = 0$. Let $x_1[0, \mu]$ and $x_2 = [1/\mu; 0]$ be two approximate solutions (the vector x_2 with error $e_2 = 1/\mu \gg 1$ can hardly be recognized as approximate solution but we shall ignore this fact). We have*

$$r_1 = \mu\sqrt{2},\ e_1 = \mu,\ r^2 = \mu^2\sqrt{2},\ e_2 = 1/\mu$$

 doi:10.20944/preprints202412.0780.v1

and

$$\frac{r_1}{r_2} = \frac{1}{\mu} \gg 1, \ \frac{e_1}{e_2} = \mu^2 \ll 1$$

This is possible since the matrix B is very badly conditioned with $\operatorname{cond}(B) = \mu^{-3} \gg 1$ *and the estimate (9) is achieved.*

Denote $T = (0,1) \subset \mathbb{R}_+$ and let $\{x(t) : t \in T\} \subset \mathbb{R}(n,1)$ be a family of approximate solutions of the equation $Ax = b$, where the function $x : T \to \mathbb{R}(n,1)$ is differentiable. Let $x(t)$ tends to x_0 as $t \to \infty$ and denote by $e(t) = \|x(t) - x_0\|$ and $r(t) = \|Ax(t) - b\|$ the error and the remainder of the approximate solution $x(t)$. These relations define the remainder r as a parametric function of e and vice versa.

Proposition 11. *There exists a smooth function $x : T \to \mathbb{R}(n,1)$ such that the following assertions hold true.*

1. *The function $e : T \to \mathbb{R}_+$ is smooth and decreasing.*
2. *The function $r : T \to \mathbb{R}_+$ is smooth and non-monotone in each interval $(0, \beta)$, where $0 < \beta < 1$.*
3. *The function $e \mapsto r(e)$ (whenever defined) is smooth and non-monotonic in each arbitrarily small subunterval $(0, \theta)$ of the interval $(0, \varepsilon)$, where $\varepsilon = \sup\{e(t) : t \in T\}$.*

Example 27. *Let $n = 2$,*

$$A = \begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix}, \ x(t) = \begin{bmatrix} \xi_1(t) \\ \xi_2(t) \end{bmatrix}, \ b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and $a \notin \{-1, 0, 1\}$. Then $x_0 = 0$ and

$$r(t) = \sqrt{a^2 \xi_1^2(t) + \xi_2^2(t)}, \ e(t) = \sqrt{\xi_1^2(t) + \xi_2^2(t)}$$

Let $t \in T$ and $\xi_1(t) = t^3 \cos(1/t)$, $\xi_2(t) = t^3 \sin(1/t)$. We have $e(t) = t^3$ and

$$r(t) = t^3 \sqrt{1 + (a^2 - 1)\cos^2(1/t)}$$

Therefore

$$r(e) = e\sqrt{1 + (a^2 - 1)\cos^2\left(e^{-1/3}\right)}$$

The differentiable function $e \mapsto r(e)$ thus defined is non-monotone on each interval $(0, \theta)$

This non-monotonicity means that for any $\theta > 0$ there exist infinitely many points $e_1, e_2 \in (0, \theta)$ with $e_1 < e_2$ and $r(e_1) > r(e_2)$. Thus the LSM may be misleading not only for linear algebraic equations but also for LLSP for which the method had been specially designed.

10. Integrals and Derivatives

10.1. Integrals

The calculation of the definite integral $I = \operatorname{int}(f, a, b)$ is one of the oldest computational problems in numerical mathematical analysis. According to [25] a variant of the trapezoid rule was used in Babylon 50 years BCE for integrating the velocity of Jupiter along the ecliptic.

Usually the integral is solved for $a, b \in \mathbb{R}$, $a < b$, but the case when one or both of the limits a, b, are infinite and the integral is improper is also considered. There are many sophisticated algorithms and computer codes for solving such integrals, see [1]. In MATLAB® definite integrals of function of one variable $f[a,b] \to \mathbb{R}$ are computed by the command `integral(f,a,b)`.

In this section we apply BFCT to solving an integral with highly oscillating integrand by the standard trapezoidal rule [26].

 doi:10.20944/preprints202412.0780.v1

Let $a = x_1 < x_2 < \cdots < x_n = b$ be a partition of $[a,b]$. Sophisticated integration schemes for computing I using values $y_k = f(x_k)$ had been developed in the pre-computer ages (before 1950) when the computation of y_k for n of order 10^3 was a problem. Now BFCT allow to obtain results for large n up to 10^8, using simple quadratures such as the formula of trapezoids with equal spacing [27] described below as MATLAB® code

```
n = 10^m; X = linspace(a,b,n+1); Y = f(X); h = (b-a)/n; ...
T_n = h*(sum(Y(2:n)) + (Y(1) + Y(n+1))/2)
```

If the function f is twice continuously differentiable then the absolute error of the trapezoidal method with equal spacing is estimated as

$$|T_n - I| \le E_n(f,a,b) = \frac{\gamma(f,a,b)}{n^2} \tag{10}$$

where

$$\gamma(f,a,b) = \frac{(b-a)^3}{12}\mu_2(f),\ \mu_2(f) = \max\{|f''(x)| : x \in [a,b]\}$$

Example 28. *Consider the integral* $I = \mathrm{int}(f,a,b) = F(b) - F(a)$, *where* $f = F'$, $a > 0$ *and*

$$F(x) = cx + x^2 \sin\left(x^{-1}\right),\ x \in [a,b]$$

Then

$$f(x) = c + 2x\sin\left(x^{-1}\right) - \cos\left(x^{-1}\right),\ f''(x) = x^{-4}\cos\left(x^{-1}\right)$$

For

$$a = \frac{1}{m\pi},\ b = \frac{1}{\pi},\ c = \frac{m\pi}{m-1},\ m \in \mathbb{Z}_2$$

we have

$$\sin\left(a^{-1}\right) = 0,\ \cos\left(a^{-1}\right) = -1,\ F(a) = ac,\ \sin\left(b^{-1}\right) = 0,\ F(b) = bc$$

and $I = c(b-a) = 1$.

For $m = 11$ *the number* $a \simeq 0.0289$ *is relatively small and the function* f *is highly oscillating close to* a, *see Figure 2. Next we have*

$$\mu_2(f) = |f''(a)| = (11\pi)^4,\ \gamma(f,a,b) = 2750\pi/3 \simeq 2880$$

and $E_n(f,a,b) = 2880\, n^{-2}$.

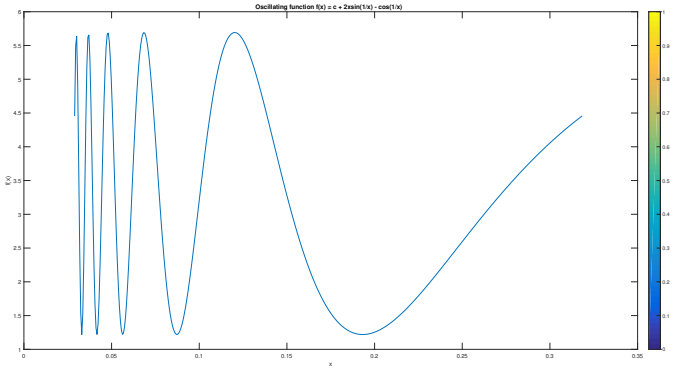

Figure 2. Oscillating function.

 doi:10.20944/preprints202412.0780.v1

The error $|T_n - 1|$ *and the ratio* $r_n(f,a,b) = |T_n - 1|/E_n(f,a,b)$ *for* $n = 10^m$ *and* m *up to* 8 *are shown in Table 4.*

Table 4. Errors and estimates for the trapezoidal rule.

n	$\lvert T_n - 1\rvert$	$r_n(f,a,b)$
5	2.7522×10^{-2}	2.3724×10^{-4}
10	9.5472×10^{-4}	3.2918×10^{-5}
100	2.0633×10^{-5}	7.1142×10^{-5}
1 000	4.3650×10^{-7}	1.5051×10^{-4}
10 000	4.3842×10^{-9}	1.5117×10^{-4}
100 000	4.3844×10^{-11}	1.5117×10^{-4}
1 000 000	4.4420×10^{-13}	1.5316×10^{-4}
1 500 000	1.9340×10^{-13}	1.5004×10^{-4}
2 000 000	1.0059×10^{-13}	1.3873×10^{-4}
5 000 000	3.4084×10^{-14}	1.5189×10^{-4}
10 000 000	1.9762×10^{-14}	6.8139×10^{-4}
50 000 000	2.1316×10^{-14}	1.8375×10^{-2}
100 000 000	5.0848×10^{-14}	1.7533×10^{-1}

For n *from* 1 000 *to* 1 000 000 *the error (which is both absolute and relative since* $I = 1$*) decreases as* $0.4\,n^{-2}$ *while the ratio of the error and the estimate* $E_n(f,a,b)$ *is approximately constant about* 1.5×10^{-4}*. The behavior of the error* $|T_n - 1|$ *is correctly evaluated but the bound* $\mu_2(f) = a^{-4}$ *for* $|f''(x)|$ *considerably overestimates the real behavior of* $|f''(x)|$ *although this bound is achieved for* $x = a$ *and is hence not improvable.*

For n *between* 1 500 000 *and* 100 000 000 *the accuracy of the computed solution does not increase with* n *and shows chaotic behavior due to accumulation of rounding errors. The result* T_n *for* $n = 1000$ *and even for* $n = 100$ *is satisfactory from practical viewpoint.*

For the choice of f, a, b as in Example 28 the MATLAB® code `integral(f,a,b)` produces the answer with error 7.1720×10^{-14}. At the same time the NLF $F(b) - F(a)$ gives the result $1 - \rho$ with error $\rho \simeq 1.1011 \times 10^{-16}$. So in this case the code `integral(f,a,b)` does not use NLF.

10.2. Derivatives

Numerical differentiation of a scalar function f of scalar or vector argument x is used in many cases, i.e. when analytical expression $f(x)$ is not available or is too complicated, when numerical algorithms are applied that use approximate derivatives f f, etc. In the simplest yet most spread case the derivative $f'(x)$ is approximated numerically by the first order finite difference $\Delta_h(f,x) = (f(x+h) - f(x))/h$, where $|h|$ is small but not too small. For given a x the behavior of $\Delta_h(f,x)$ as function of h resembles a saw, see Example 29 below.

Example 29. *Consider the first difference* $\Delta_h(x,1)$ *for the simplest non-constant function* $f(x) = x$ *at the point* $x = 1$*. We have* $\Delta_h(x,1) = (1 + h - 1)h^{-1}$*. The graph of the computed function* $h \mapsto (1 + h - 1)h^{-1}$ *for* $h \in [-3\text{eps}, 3\text{eps}]\backslash\{0\}$ *is shown at Figure 3. The graph of the exact function* $h \to 1$ *is the straight line in red.*

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 December 2024 doi:10.20944/preprints202412.0780.v1

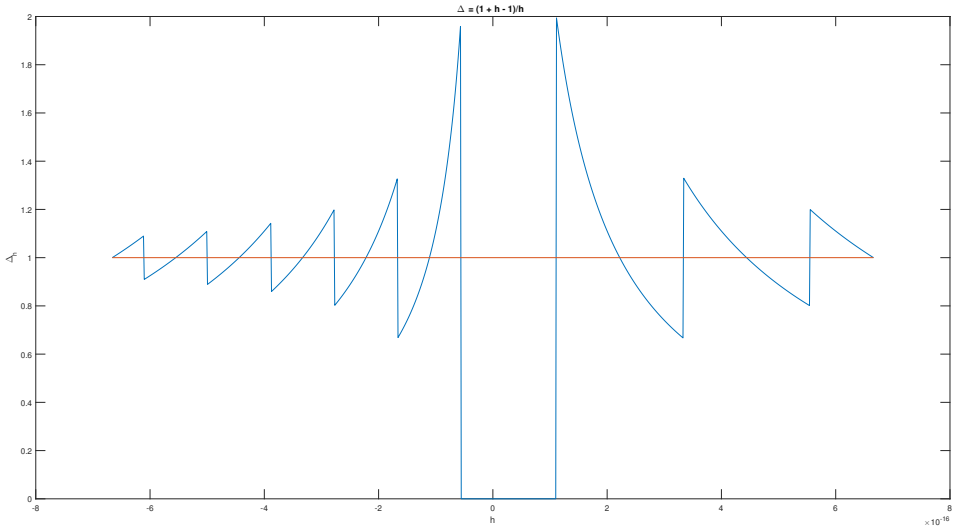

Figure 3. Computed first difference of the function $f(x) = x$.

Let $f'(x) \neq 0$ and $x \neq 0$. When h/x is too small, e.g. $h/x \in [-\rho/2, \rho]$, the computed value of $f(x+h)$ is $f(x)$, $\Delta_h(f,x) = 0$ and the derivative $f'(x)$ is computed as 0 with 100% relative error. This numerical catastrophe may be avoided by BFCT revealing the behavior of $\Delta_h(f,x)$ for small values of h.

The general principle here is that h must be chosen as $C(\text{eps})^{1/2} = C/10^8$, and the problem is how to estimate the constant C especially in RTA. When no information for a constant $C > 0$ is available, the heuristic rule is to set $C = 1$.

11. VPA versus FPA

There are many types of variable precision arithmetics (VPA) and computer languages in which the VPA are realized. Here we include systems using symbolic objects like π, $\sqrt{2}$, etc. VPA may work with very high precision (actually, with infinite precision) but the performance of numerical algorithms with VPA may be slow and not suitable for RTA. In contrast, binary double-precision floating point arithmetic (FPA) works with a finite set of machine numbers, has finite precision and works relatively fast. Thus FPA is suitable for RTA.

12. Conclusion

The combination of BFCT and CPRS provides a powerful tool for solving problems and checking computed solutions and a priori accuracy estimates for many computational problems. Also, according to the authors, BFCT may become a useful complement in the implementation of techniques of artificial intelligence (AI) to the systems for doing mathematics such as MATLAB® [6,7] and Maple [8].

Our main results are summarized in Propositions 1, 2, 3, 4, 5, 6 and 19. Most of the results presented are valid for using BFCT in FPA. Using extended precision as in VPA, some of the observed effects may no more occur. This may be done at the price of considerable increase of the computational time thus excluding some RTA.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets generated during the current study are available from the author upon reasonable request.

 doi:10.20944/preprints202412.0780.v1

Conflicts of Interest: The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Stoer, J., Bulirsch, R. *Introduction to Numerical Analysis*; Springer New York: New York, 2002; ISBN 978-0387954523, doi:10.1007/978-0387217383.
2. Faires, D., Burden, A. *Numerical Analysis*, Cengage Learning: Boston, 2016; ISBN 978-1305253667.
3. Chaptra, S. *Applied Numerical Methods with MATLAB for Engineers and Scientists* (5th edition), McGraw Hill: 2017; ISBN 978-12644162604.
4. Novak, K. *Numerical Methods for Scientific Computing*, Equal Share Press: 2022; ISBN 978-8985421804.
5. Driscoll, T., Braun, R. *Fundamentals of Numerical Computations*, SIAM: 2022; doi:10.1137/1.9781611975086.
6. TheMathWorks, Inc. *MATLAB Version 9.9.0.1538559 (R2020b)*; The MathWorks, Inc.: Natick, MA, USA, 2020. Avaliable online: www.mathworks.com
7. D. Higham, N. Higham. *MATLAB Guide* (3rd edition), SIAM, Philadelphia, 2017; ISBN 978-1611974652.
8. Maplesoft. *Maple 2017.3*, Ontario, 2017. Avaliable online: www.maplesoft.com/ products/maple/
9. Mathematica. Avaliable online: www.wolfram.com/mathematica/
10. WolframAlpha. Avaliable online: www.wolframalpha.com/
11. Borbu, A., Zhu, S. *Monte Carlo Methods*, Springer Singapore, 2020; ISBN 978-9811329708, doi:10.1007/9789811319715.
12. Davies, A. et al. Advancing mathematics by guiding human intuition with AI. *Nature* **600** (2021); doi:10.1038/s4158602104086x
13. Fink, T. Why mathematics is set to be revolutionized by AI, *Nature*, **629** (2024); doi:10.1038/d4158602401413w.
14. IEEE Computer Society, *754-2019–IEEE Standard for Floating-Point Arithmetic*, IEEE, NJ, 2019; doi:10.1109/IEEESTD.2019.8766229.
15. Higham, N. *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 2002; ISBN 0-898715210.
16. Higham, N., Konstantinov, M., Mehrmann, V., Petkov, P. The sensitivity of computational control problems, *IEEE Control Systems Magazine*, **24** (2004), 28-43; doi:10.1109/MCS.2004.1272744.
17. Goldberg, D. What every computer scientist should know about floating-point arithmetic, *ACM Computing Surveys*, **23** (1991), no. 1, 5-48; doi:10.1145/103162.103163.
18. Pugh, C. *Real Mathematical Analysis*, Springer Nature, Switzerland AG, 2015; ISBN 978-3319177700, doi:10.1007/978-3319177717
19. Konstantinov, M., Petkov, P. Computational errors, *International Journal of Applied Mathematics*, **35** (2022), 1, 181-203; doi:10.12732/ijam.v35i1.14.
20. Balinski, M., Young, H. *Fair Representation: Meeting the Ideal One Man One Vote*, Brookings Institutional Press (2nd edition), New Haven and London, 2001; ISBN 978-0815701118, doi:10.2307/1288706.
21. Booth, A. *Numerical Methods* (3rd edition), London, Butterworths Scientific Publications , London, 1966; LCCN 57004892.
22. Konstantinov, M, Petkov, P. Remainders vs. errors, *AIP Conference Proceedings*, **1789** (2016), 060007; doi:10.1063/1.4968499.
23. Lagarias, J., Reeds, J., Wright, M., Wright, P. Convergence properties of the Nelder-Mead simplex method in low dimensions, *SIAM Journal on Optimization*, **9** (1998), no. 1, 112-147; doi:10.1137/S1052623496303470.
24. Nelder, J., Mead, R. A simplex method for function minimization, *Computer Journal*, **7** (1965), no. 4, 308-313; doi:10.1093/comjnl/7.4.308.
25. Ossendrijver, M. Ancient Babylonian astronomers calculated Jupiter's position from the area under a time-velocity graph, *Science*, **351** (2016), 6272, 482-484; doi:10.1126/science.aad8085.
26. Rahman, Q.; Schmeisser, G. Characterization of the speed of convergence of the trapezoidal rule, *Numerische Mathematik* **1990**, 57, 123-138; doi:10.1007/BF01386402.
27. Atkinson, K. *An Introduction to Numerical Analysis* (2nd edition); John Wiley & Sons: New York, NY, USA, 1989; ISBN 978-0471500230.
28. Glazman, M.; Ljubich, J. *Finite-Dimensional Linear Analysis: A Systematic Presentation in Problem Form* (Dover Books in Mathematics); Dover Publications: New York, NY, USA, 2006; ISBN 978-04866453323.
29. Jackobson, N. *Basic Algebra I*; W. Freeman & Co Ltd.: London, UK, 1986, ISBN 978-07167114804

 doi:10.20944/preprints202412.0780.v1

30. Andrews, G. *The Theory of Partitions* (revisited edition); Cambridge University Press: Cambridge, 2008; ISBN 978-0521637664.
31. Konstantinov, M.; Petkov, P. On Schur forms for matrices with simple eigenvalues; *Axioms* **2024**, 13, 12, 839; doi.org/10.3390/axioms13120839.