The Chinese University of Hong Kong, Shenzhen
SDS · School of Data Science
Andre Milzarek · Fall Semester 2024-25

DDA 3005 — Numerical Methods

Solutions 1

Problem 1 (Floating-Point Numbers and Sample Variance): (approx. *25* points)
In this exercise, we want to compute the (unbiased) sample variance of a sequence of numbers $\{x_i\}_{i=1,\dots,n}$ using floating-point arithmetics. The sample variance can be calculated via one of the equivalent formulas

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \quad (\mathsf{A}) \qquad \text{or} \qquad \sigma^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - n\bar{x}^2\right] \quad (\mathsf{B}),$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ denotes the associated sample mean.

Here, we consider the numbers $x_1 = 2021$, $x_2 = 2022$, $x_3 = 2023$, and $x_4 = 2024$ with $n = 4$. Calculate the sample variance σ^2 via the two formulas (A) and (B) for $\{x_i\}_{i=1,\dots,4}$ using a decimal, normalized floating-point system with $\beta = 10$, $p = 6$, $U = -16$, $L = 16$, and rounding to nearest. You can assume that arithmetic floating-point operations are executed in order and that square operations have priority, i.e.,

$$\sum_{i=1}^{4} x_i \approx ((x_1 \oplus x_2) \oplus x_3) \oplus x_4, \quad n\bar{x}^2 \approx n \odot (\bar{x} \odot \bar{x}), \quad \dots \quad \text{etc.}$$

Provide detailed calculations and steps that illustrate your derivation and results.

Solution : In the floating-point system, the numbers x_1, x_2, x_3 and x_4 are represented as

$$x_1 = 2.02100 \cdot 10^3,\ x_2 = 2.02200 \cdot 10^3,\ x_3 = 2.02300 \cdot 10^3,\ x_4 = 2.02400 \cdot 10^3,$$

and we obtain $\bar{x} = 8.09000 \cdot 10^3 \oslash 4.00000 \cdot 10^0 = 2.02250 \cdot 10^3\ (= 2022.5)$. (No information is lost when calculating $\bar{x}$ as all involved numbers are machine numbers). We first calculate σ^2 according to (A):

$$\begin{aligned}
(x_1 \ominus \bar{x})^2 &= (-1.50000 \cdot 10^0)^2 = 2.25000 \cdot 10^0,\\
(x_2 \ominus \bar{x})^2 &= (-5.00000 \cdot 10^{-1})^2 = 2.50000 \cdot 10^{-1}\\
(x_3 \ominus \bar{x})^2 &= (5.00000 \cdot 10^{-1})^2 = 2.50000 \cdot 10^{-1},\\
(x_4 \ominus \bar{x})^2 &= (1.50000 \cdot 10^0)^2 = 2.25000 \cdot 10^0.
\end{aligned}$$

Summing these numbers results in $\sum_{i=1}^{4}(x_i - \bar{x})^2 = 5.00000 \cdot 10^0$ (no information is lost) and $\sigma^2 = 5.00000 \cdot 10^0 \oslash 3.00000 \cdot 10^0 = 1.66667 \cdot 10^0$. We proceed with formula (B):

$$x_1^2 = 4.08444 \cdot 10^6,\ x_2^2 = 4.08848 \cdot 10^6,\ x_3^2 = 4.09253 \cdot 10^6,\ x_4^2 = 4.09658 \cdot 10^6,$$

and $\bar{x}^2 = 4.09051 \cdot 10^6$. We obtain $\sum_{i=1}^4 x_i^2 = (8.17292 \cdot 10^6 \oplus x_3^2) \oplus x_4^2 = \mathrm{fl}(1.226545 \cdot 10^7) \oplus x_4^2 = 1.22654 \cdot 10^7 \oplus 4.09658 \cdot 10^6 = 1.63620 \cdot 10^7$ (notice that there is a tie in the second operation and we round down to obtain an even last digit) and

$$\sum_{i=1}^{4} x_i^2 \ominus (4 \odot \bar{x}^2) = 1.63620 \cdot 10^7 \ominus 1.63620 \cdot 10^7 = 0.00000 \cdot 10^0 \ (= 0).$$

This yields $\sigma^2 = 0.00000 \cdot 10^0 \oslash 3.00000 \cdot 10^0 = 0.00000 \cdot 10^0 = 0$. This is of course significantly different from the (true) result in (A). In particular, the sample variance can not be zero in this case! Hence, the cancellation effect in (B) can be much more severe.

Problem 2 (Plotting Polynomials): (approx. *25* points)
We consider the polynomial function $p : \mathbb{R} \to \mathbb{R}$, $p(x) = (x-3)^8$. The polynomial p has the explicit representation

$$p(x) = x^8 - 24x^7 + 252x^6 - 1512x^5 + 5670x^4 - 13608x^3 + 20412x^2 - 17496x + 6561. \quad (1)$$

a) Plot the function p for $x = 2.920, 2.921, \ldots, 3.080$ using the representation shown in (1). Discuss whether your plot is an accurate representation of the mapping p.

b) Recreate the plot — now using the compact formula $p(x) = (x-3)^8$.

Compare the results obtained in part a) and b) and discuss your observations. Can you explain the observed effects?

Solution : The two plots differ significantly and are shown below in Figure 1.

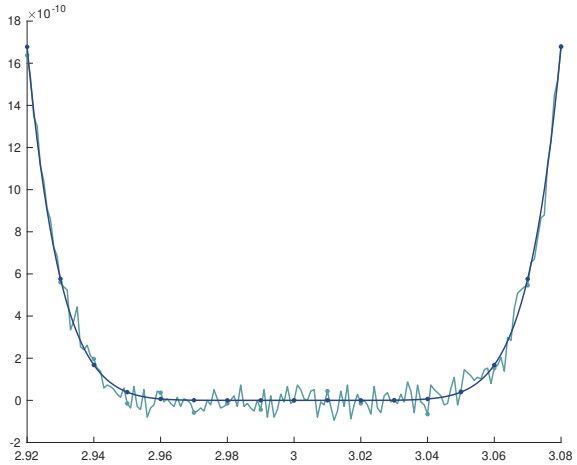

Figure 1: Plot of the polynomial $p(x)$ using the representation (1) (•) and the compact formula $(x-3)^8$ (•).

Specifically, the plot using the explicit representation (1) has a very different behavior and does not seem to depict a polynomial function (we expect $p(x) = (x-3)^8$ to "look much smoother"!). In addition, the representation (1) sometimes causes results with wrong signs (which corresponds to an error of more than 100%). This distinctive behavior can be explained by the cancellation effects occurring in the representation (1): we add and subtract large numbers of the order $3^8 \approx 6500$ to obtain a result that is of order 10^{-9}. To be more precise, let us consider the magnitudes of each of the summands separately; we have:

x	x^8	$-24x^7$	$252x^6$	$-1512x^5$	$5670x^4$	$-13608x^3$	$20412x^2$	$-17496x$	6561
2	6561	-52488	183708	-367416	459270	-367416	183708	-52488	6561

Thus, the largest summand has magnitude $\approx 2^{19}$. Using `double precision` (or `float64`), the machine precision is given by $\varepsilon_{\text{mach}} \approx 2^{-52}$. Consequently, the accuracy of the arithmetic operations and summations in (1) is bounded by $2^{19} \cdot 2^{-52} = 2^{-33} \approx 1.2 \cdot 10^{-10}$ (i.e., adding/subtracting a number that is smaller than 2^{-33} would not have any effect). We can illustrate this more clearly by considering the computed results:

x	$10^9 \cdot p(x)$ via (1)	$10^8 \cdot (x-3)^8$	x	$10^9 \cdot p(x)$ via (1)	$10^9 \cdot (x-2)^9$
2.92	0.163709046319127	0.167772160000001	3.00	0	0
2.93	0.056024873629212	0.057648010000002	3.01	0.004365574568510	0.000000010000000
2.94	0.019645085558295	0.01679616000000	3.02	-0.001455191522837	0.000002560000000
2.95	-0.0014551915228379	0.003906250000000	3.03	0	0.000065610000000
2.96	0.003637978807092	0.000655360000000	3.04	-0.006548361852765	0.000655360000000
2.97	-0.005820766091347	0.000065610000000	3.05	0.004365574568510	0.003906250000000
2.98	-0.001455191522837	0.000002560000000	3.06	0.015279510989785	0.016796160000000
2.99	-0.004365574568510	0.000000010000000	3.07	0.054569682106376	0.057648010000002
			3.08	0.168074620887637	0.167772160000001

The absolute error varies around 10^{-11}. However, for $x = 2.99$ and $x = 3.01$, we obtain a relative error of approximately $\approx 5 \cdot 10^5$! Hence, with these large error margins, the differences in Figure 1 are not surprising.

Problem 3 (A "Shaky" Recursion): (approx. *25* points)

We consider the recursion

$$x_{k+1} = \frac{9}{4}x_k - \frac{1}{2}x_{k-1} \quad k = 2, 3, \ldots \tag{2}$$

with initial values $x_1 = \frac{1}{3}$ and $x_2 = \frac{1}{12}$.

a) Write a `MATLAB` or `Python` program to compute the first 60 elements of the sequence $\{x_k\}_{k=1,\ldots,60}$ using (2). Create a semilog plot of the obtained values as a function of k.

b) The exact solution to the difference recursion (2) is given by $x_k = \frac{1}{3}4^{1-k}$. Plot the mapping $k \mapsto \frac{1}{3}4^{1-k}$ for $k = 1, \ldots, 60$ and compare the behavior of the obtained graph with the results from part a). What is your observation?

c) Next, let us consider the recursion (2) with the general initial points $x_1 = a$ and $x_2 = b$, where $a, b \in \mathbb{R}$. Show that the general solution to this problem is given by

$$x_k = \frac{2^{k-1}}{7}(4b - a) + \frac{4^{2-k}}{7}(2a - b) \tag{3}$$

(e.g., through induction over k).

d) Can you explain the numerical effects and behavior observed in part a) and b)?

⋆e) Can you provide a direct derivation of the (mysterious) formula (3) from the recursion (2) (with $x_1 = a$ and $x_2 = b$), i.e., without knowing the specific form of (3) already?

Remark: Part e) is a bonus question and you do not need to submit solutions for this part in order to receive full marks (25 points) on this problem. A fully correct solution to part e) can count up to additional *10 points*. However, the maximum number of achievable points for this assignment sheet is still capped at 100 points.

Solution : The `MATLAB` code for parts a) and b) of this exercise are shown below. Running the code, we obtain the output (both in `MATLAB` and `Python`) shown in Figure 2.

The behavior of the recursively computed iterates $\{x_k\}_k$ is significantly different from the true solutions $x_k = \frac{1}{3}4^{1-k}$! Indeed, for $k > 20$, the computed iterates (following (2)) seem to jump from monotonic convergence to 0 (which was very close to the behavior of the true solutions $\frac{1}{3}4^{1-k}$, $k \in \mathbb{N}$) to monotonic divergence to ∞.

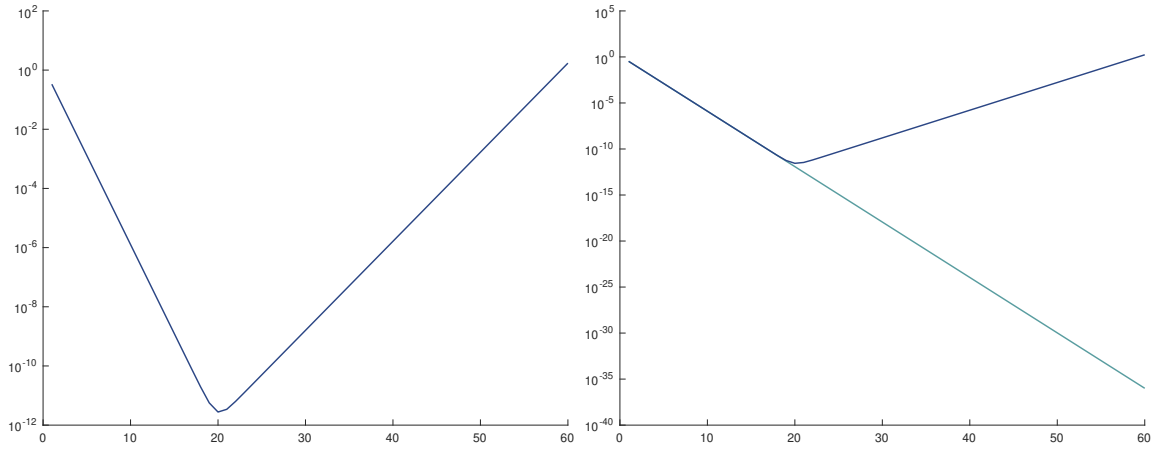

Figure 2: Left: Plot of the sequence $\{x_k\}_k$ using the recursive formula (2). Right: Plot of the sequence $\{x_k\}_k$ from part a) (in blue •) and the true solutions $\{\frac{1}{3}4^{1-k}\}_k$ (in green •).

```
n = 60;
x = zeros(n,1);

% Compute x_k recursively

x(1) = 1/3;
x(2) = 1/12;

for i = 3:n
    x(i) = 9*x(i-1)/4 - x(i-2)/2;
end

% Plot results

c_blue  = [43,70,133]/255;
c_turq  = [89,157,160]/255;

figure;
hold on

plot(1:n,log10(x),'Color',c_blue,'LineWidth',1.2);
ytickformat('10^{%g}');

hold off
saveas(gcf,strcat('exercise_a3_1.eps'),'epsc');

figure;
hold on

plot(1:n,log10(4.^(0:-1:1-n)/3),'Color',c_turq,'LineWidth',1.2);
```

```
plot(1:n,log10(x),'Color',c_blue,'LineWidth',1.2);
ytickformat('10^{%g}');

hold off
saveas(gcf,strcat('exercise_a3_2.eps'),'epsc');
```

The `Python` code for parts a) and b) is similar to the presented `MATLAB` code and can be found in the attachment.

We now continue with part c). We prove (3) via induction. For $k = 1$, it holds that $x_1 = \frac{1}{7}(4b - a) + \frac{4}{7}(2a - b) = a$ and $x_2 = \frac{2}{7}(4b - a) + \frac{1}{7}(2a - b) = b$. Hence, the base cases $k = 1, 2$ are correct. Let us assume now that (3) holds for some $k = 1, \dots, i \in \mathbb{N}$; we have

$$\begin{aligned} x_{i+1} = \frac{9}{4}x_i - \frac{1}{2}x_{i-1} &= \frac{9}{4}\frac{2^{i-1}}{7}(4b - a) + \frac{9}{4}\frac{4^{2-i}}{7}(2a - b) - \frac{1}{2}\frac{2^{i-2}}{7}(4b - a) - \frac{1}{2}\frac{4^{3-i}}{7}(2a - b) \\ &= \frac{1}{7}\left[\frac{9}{4}2^{i-1} - \frac{1}{2}2^{i-2}\right](4b - a) + \frac{1}{7}\left[\frac{9}{4}4^{2-i} - \frac{1}{2}4^{3-i}\right](2a - b) \\ &= \frac{2^{i-3}}{7}[9 - 1](4b - a) + \frac{4^{1-i}}{7}[9 - 8](2a - b) = \frac{2^i}{7}(4b - a) + \frac{4^{1-i}}{7}(2a - b), \end{aligned}$$

which is (3) for $k = i + 1$. Hence, by induction, (3) is correct for all $k \in \mathbb{N}$.

Part d): The mathematical insights from part c) allow us to explain the numerical observations made in part a) and b). Using $a = \frac{1}{3}$ and $b = \frac{1}{12}$, we see that the general formula (3) exactly reduces to $x_k = \frac{1}{3}4^{1-k}$. Furthermore, if $a \neq 4b$, then the general formula contains the term 2^{k-1} which will eventually dominate and thus, $x_k \to \infty$.

Computing x_k via the recursion (2) is affected by rounding errors and cancellation. These errors eventually cause x_k and x_{k-1} to no longer satisfy $x_{k-1} = 4x_k$, i.e., $a = x_{k-1}$ and $b = x_k$ will play the role of "new initial points" of the recursion leading to a completely different behavior and solution. Hence, the rounding errors cause the iterates to eventually shift to a different solution.

`iter`	$4x_k - x_{k-1}$	`iter`	$4x_k - x_{k-1}$
2	0	15	0.000000000000171
3	0.000000000000000	16	0.000000000000341
4	0.000000000000000	17	0.000000000000682
5	0.000000000000000	18	0.000000000001364
6	0.000000000000000	19	0.000000000002728
7	0.000000000000001	20	0.000000000005457
8	0.000000000000001	21	0.000000000010914
9	0.000000000000003	22	0.000000000021828
10	0.000000000000005	23	0.000000000043656
11	0.000000000000011	24	0.000000000087311
12	0.000000000000021	25	0.000000000174623
13	0.000000000000043	26	0.000000000349246
14	0.000000000000085	27	0.000000000698492

Part e): To derive (3), we notice that the recursion (2) can be written as

$$\boldsymbol{y}^{k+1} := \begin{pmatrix} x_{k+1} \\ x_k \end{pmatrix} = \begin{pmatrix} \frac{9}{4} & -\frac{1}{2} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_k \\ x_{k-1} \end{pmatrix} = \begin{pmatrix} \frac{9}{4} & -\frac{1}{2} \\ 1 & 0 \end{pmatrix} \boldsymbol{y}^k =: \boldsymbol{A}\boldsymbol{y}^k.$$

Hence, unrolling the recursion, this yields $\boldsymbol{y}^{k+1} = \boldsymbol{A}^{k-1}\boldsymbol{y}^2$. In order to obtain an explicit representation for $\boldsymbol{y}^{k+1}$, we need to compute $\boldsymbol{A}^{k-1}$. This can be done via an eigenvalue decomposition of $\boldsymbol{A}$. It holds that

$$\det(\boldsymbol{A} - \lambda \boldsymbol{I}) = -\lambda\left(\frac{9}{4} - \lambda\right) + \frac{1}{2} = \lambda^2 - \frac{9}{4}\lambda + \frac{1}{2}.$$

Thus, the eigenvalues of $\boldsymbol{A}$ are given by $\lambda_{1/2} = \frac{1}{2}(\frac{9}{4} \pm \sqrt{\frac{81}{16} - 2}) = \cdots = 2 \,/\, \frac{1}{4}$. The corresponding (not normalized) eigenvectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are given by:

$$\boldsymbol{v}_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{v}_2 = \begin{pmatrix} 1 \\ 4 \end{pmatrix}.$$

Setting $\boldsymbol{V} = \begin{pmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 \end{pmatrix}$, we further have

$$\boldsymbol{V}^{-1} = \frac{1}{7}\begin{pmatrix} 4 & -1 \\ -1 & 2 \end{pmatrix} \quad \text{and} \quad \boldsymbol{A} = \boldsymbol{V}\begin{pmatrix} 2 & \\ & \frac{1}{4} \end{pmatrix}\boldsymbol{V}^{-1}.$$

(We will review eigendecompositions and these techniques in detail in later parts of the course).

We now obtain

$$\boldsymbol{A}^{k-1} = \boldsymbol{V}\begin{pmatrix} 2^{k-1} & \\ & 4^{1-k} \end{pmatrix}\boldsymbol{V}^{-1}$$

and

$$\begin{aligned} \boldsymbol{A}^{k-1}\boldsymbol{y}^2 &= \frac{1}{7}\boldsymbol{V}\begin{pmatrix} 2^{k-1} & \\ & 4^{1-k} \end{pmatrix}\begin{pmatrix} 4 & -1 \\ -1 & 2 \end{pmatrix}\begin{pmatrix} b \\ a \end{pmatrix} = \frac{1}{7}\begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix}\begin{pmatrix} 2^{k-1}(4b-a) \\ 4^{1-k}(2a-b) \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{7}2^k(4b-a) + \frac{1}{7}4^{1-k}(2a-b) \\ \frac{1}{7}2^{k-1}(4b-a) + \frac{1}{7}4^{2-k}(2a-b) \end{pmatrix}. \end{aligned}$$

This finishes the proof and derivation. (There are, of course, also different ways to derive this recursion).

Problem 4 (Squares and Error Analysis): (approx. *25* points)

In this problem, we want to analyze numerical errors when calculating the difference of two square numbers. Specifically, for given $a, b \in \mathbb{R}$, we consider the function

$$y = f(a, b) := a^2 - b^2$$

and the associated floating-point approximation

$$\hat{y} = \hat{f}(a, b) := \mathrm{fl}(f(a, b)) = (a \odot a) \ominus (b \odot b).$$

Throughout this exercise and to simplify the analysis, we assume that a, b are machine numbers.

a) Show that the absolute forward error (total error) can be estimated via

$$|\hat{y} - y| \leq \mathcal{O}(\varepsilon_{\text{mach}}) \max\{a^2, b^2\}$$

(for a sufficiently small machine precision $\varepsilon_{\text{mach}}$).

– You can assume that the underlying floating-point system satisfies the `IEEE`-standard 754, i.e., we have $u \circledast v = \mathrm{fl}(u * v) = (1 + \varepsilon)(u * v)$ for all machine numbers u, v, every arithmetic operation $* \in \{+, -, \cdot, /, \sqrt{\cdot}\}$, and some ε with $|\varepsilon| \leq \varepsilon_{\text{mach}}$.

b) Consider a decimal, normalized floating-point system with $\beta = 10$, $p = 4$, $U = -9$, $L = 9$, and rounding to nearest. Construct an example and choose a, b such that

$$\frac{|\hat{y} - y|}{|y|} > \frac{1}{2}.$$

c) Can you design an alternative (more accurate) algorithm $\tilde{y} = \tilde{f}(a, b)$ satisfying

$$\frac{|\tilde{y} - y|}{|y|} = \frac{|\tilde{f}(a, b) - f(a, b)|}{|f(a, b)|} \approx \mathcal{O}(\varepsilon_{\text{mach}})$$

for all machine numbers a, b? Provide detailed explanations! Repeat the calculations for the example in part b) using the alternative algorithm $\tilde{f}$. What are your observations?

Solution :

a) We assume that a, b are floating-point numbers. Then, using the `IEEE`-standard, there exist $\varepsilon_1, \varepsilon_2, \varepsilon_3$ with $|\varepsilon_i| \leq \varepsilon_{\text{mach}}$ such that $(a \odot a) = (1 + \varepsilon_1)a^2$, $(b \odot b) = (1 + \varepsilon_2)b^2$, and

$$(a \odot a) \ominus (b \odot b) = (1 + \varepsilon_3)[a^2 - b^2 + \varepsilon_1 a^2 - \varepsilon_2 b^2].$$

Hence, we can decompose $\hat{y}$ as follows:

$$\hat{y} = y + \varepsilon_1 a^2 - \varepsilon_2 b^2 + \varepsilon_3[\varepsilon_1 a^2 - \varepsilon_2 b^2]$$

and using the triangle inequality, this implies

$$|\hat{y} - y| \leq \varepsilon_{\text{mach}}(a^2 + b^2) + \varepsilon_{\text{mach}}^2(a^2 + b^2) \leq 2\varepsilon_{\text{mach}}(1 + \varepsilon_{\text{mach}}) \max\{a^2, b^2\}.$$

Utilizing $2\varepsilon_{\text{mach}}(1 + \varepsilon_{\text{mach}}) = \mathcal{O}(\varepsilon_{\text{mach}})$, this finishes the proof.

b) As an example, we can use $a = 3.163$ and $b = 3.162$. We then have $a \odot a = 1.000 \cdot 10^1$ and $b \odot b = 9.998$. This yields

$$y = 6.325 \cdot 10^{-3}, \quad \text{and} \quad \hat{y} = (a \odot a) \ominus (b \odot b) = 2 \cdot 10^{-3},$$

and $|y - \hat{y}| = 4.325 \cdot 10^{-3}$. Hence, the overall forward error is given by $|y - \hat{y}|/|y| = 4.325/6.325 > 2/3 > 1/2$. There are many other possible examples (the easiest way to construct a and b is to enforce rounding errors in $a \odot a$ and $b \odot b$ that are not cancelled by the subtraction).

c) We consider the alternative algorithm

$$\tilde{y} = (a \oplus b) \odot (a \ominus b).$$

Using the `IEEE`-standard, there exist $\tilde{\varepsilon}_1, \tilde{\varepsilon}_2, \tilde{\varepsilon}_3$ with $|\tilde{\varepsilon}_i| \leq \varepsilon_{\text{mach}}$ such that

$$\tilde{y} = [(1 + \tilde{\varepsilon}_1)(a + b)] \odot [(1 + \tilde{\varepsilon}_2)(a - b)] = (1 + \tilde{\varepsilon}_1)(1 + \tilde{\varepsilon}_2)(1 + \tilde{\varepsilon}_3)(a^2 - b^2).$$

Moreover, we have $(1 + \tilde{\varepsilon}_1)(1 + \tilde{\varepsilon}_2)(1 + \tilde{\varepsilon}_3) = (1 + \tilde{\varepsilon}_1 + \tilde{\varepsilon}_2 + \tilde{\varepsilon}_1\tilde{\varepsilon}_2)(1 + \tilde{\varepsilon}_3) = 1 + \tilde{\varepsilon}_1 + \tilde{\varepsilon}_2 + \tilde{\varepsilon}_3 + \tilde{\varepsilon}_1\tilde{\varepsilon}_2 + \tilde{\varepsilon}_1\tilde{\varepsilon}_3 + \tilde{\varepsilon}_2\tilde{\varepsilon}_3 + \tilde{\varepsilon}_1\tilde{\varepsilon}_2\tilde{\varepsilon}_3$. This implies

$$|\tilde{y} - y| \leq (3\varepsilon_{\text{mach}} + 3\varepsilon_{\text{mach}}^2 + \varepsilon_{\text{mach}}^3)|a^2 - b^2|$$

and $|\tilde{y} - y|/|y| \leq 3\varepsilon_{\text{mach}} + 3\varepsilon_{\text{mach}}^2 + \varepsilon_{\text{mach}}^3 = \mathcal{O}(\varepsilon_{\text{mach}})$. Hence, the algorithm $\tilde{y} = \tilde{f}(a, b)$ is supposed to be more accurate.

Indeed, repeating the calculations for the example in part b), we obtain

$$\tilde{y} = (3.163 \oplus 3.162) \odot (3.163 \ominus 3.162) = 6.325 \cdot 10^0 \odot 1.000 \cdot 10^{-3} = 6.325 \cdot 10^{-3} = y.$$

No rounding error occurred in this computation, i.e., the result is exact.