

24.1 Floating Point and Backward Error

Chapter 16 ended with a number and a warning. The number was $\kappa(\mathbf{A})$, which bounds how much a relative error in the data can be magnified on the way to the answer. The warning was that κ says nothing about arithmetic: it “describes a problem”, and an error bound of its shape “only becomes a statement about a computation once one knows how large a perturbation the computation itself introduces”. That chapter closed the door with a sentence we now have to walk through: “Conditioning bounds what *any* method can promise; whether a given method keeps that promise is stability, and it is Chapter 24’s subject.”

This section builds the two things that were missing. First a **model** of machine arithmetic — an assumption, stated plainly as one, because floating-point arithmetic is not a theorem of linear algebra. Then the vocabulary that turns the model into verdicts about algorithms: forward error, backward error, and stability.

24.1.1 Two answers to one question

Chapter 16 §08 left a small experiment on the reader’s desk. The system was

$$\begin{aligned} 10^{-15}x_1 + x_2 &= 1, \\ x_1 + x_2 &= 2, \end{aligned}$$

whose coefficient matrix has $\kappa_\infty < 5$: by every measure of Chapter 16 this problem is as well behaved as a problem can be, and its exact solution is $(1, 1)$ to fourteen places. Eliminating x_1 with the first equation as the pivot row, and rounding to twelve significant decimal digits at every step, returns $(0, 1)$. Exchanging the two equations first, and rounding in exactly the same way, returns $(1, 1)$.

Two algorithms, one problem, one arithmetic, and one of the two answers has no correct digit in it. Nothing in Chapter 16 can tell the two apart, because κ never mentions a method. Section 2 works that example in the model below and says exactly what the exchange achieved. This section builds the language it will be said in.

24.1.2 A model of machine arithmetic

A machine holds finitely many numbers. It cannot hold $1/3$, it cannot hold $\sqrt{2}$, and between any two numbers it does hold there are infinitely many it does not. So the result of an arithmetic operation on two stored numbers is usually not a stored number, and the machine must replace it by a neighbor.

We could try to describe which numbers a particular machine stores and exactly how it rounds. That would be a description of a machine, not mathematics, and it would be obsolete before the ink dried. What is wanted instead is the *one property* of rounding that every later argument uses, elevated into an assumption.

Every operation is performed exactly and then moved to the nearest stored number, so each operation multiplies its exact answer by something within u of 1.

Definition 24.1.1. The floating-point model

A **floating-point model** consists of a finite set $\mathcal{R} \subseteq \mathbb{R}$ of **representable numbers** with $0, 1 \in \mathcal{R}$ and $-\mathcal{R} = \mathcal{R}$, a map $\text{fl}: \mathbb{R} \rightarrow \mathcal{R}$ sending each real number to a **nearest** element of $\mathcal{R}$, and a real number $u > 0$ called the **unit roundoff**, subject to the following.

- (FP1) (**Rounding.**) For every $t \in \mathbb{R}$ in range, $\text{fl}(t) = t(1 + \delta)$ for some δ with $|\delta| \leq u$.
- (FP2) (**Arithmetic.**) For $x, y \in \mathcal{R}$ and $\odot \in \{+, -, \times, \div\}$ with $y \neq 0$ when $\odot$ is $\div$, the computed value of $x \odot y$ is $\text{fl}(x \odot y)$; the computed value of $\sqrt{x}$ for $x \in \mathcal{R}$, $x \geq 0$, is $\text{fl}(\sqrt{x})$. Hence
- $$\text{fl}(x \odot y) = (x \odot y)(1 + \delta), \quad |\delta| \leq u,$$
- and likewise $\text{fl}(\sqrt{x}) = \sqrt{x}(1 + \delta)$ with $|\delta| \leq u$.
- (FP3) (**No overflow or underflow.**) Every quantity occurring in a computation we analyze is assumed to be **in range**, so that (FP1) applies to it.

Clause by clause. “A nearest element” is a mild requirement, and the only use we make of it is the observation that $|t| \leq 1$ forces $|\text{fl}(t)| \leq 1$, because $1 \in \mathcal{R}$. (FP1) is the whole content of the word *roundoff*: the error made in storing a real number is **relative**, of size at most u , and that is essentially the only thing we will ever know about fl . (FP2) says an operation is as accurate as storing its answer would be — the machine does not do anything stupid in between. The displayed identity is what every proof in this chapter actually uses, and it is worth reading aloud: *the computed product is the exact product times $1 + \delta$* . (FP3) is an honest exclusion. Numbers too large or too small for $\mathcal{R}$ break (FP1) completely, and analyzing them is a separate subject; we assume the problem away and say so each time it matters.

This is a definition, not a theorem. Nothing in Chapters 1 to 21 implies it, and nothing in this chapter proves it. It is an idealization of a machine, in the way that a frictionless plane is an idealization of a table. The arithmetic in common use satisfies it for the four operations and the square root, with u a little over 10^{-16} ; a decimal machine carrying t significant digits satisfies it with $u = \frac{1}{2} \cdot 10^{1-t}$. Every statement in this chapter that mentions rounding is a theorem **about the model**, and carries exactly as much authority as the model does.

Example. The model we compute in

Throughout this chapter, a worked number comes from the **six-digit decimal model**: $\mathcal{R}$ is the set of numbers $\pm d_1.d_2d_3d_4d_5d_6 \times 10^e$ with digits d_i , $d_1 \neq 0$ and e in some fixed finite range, together with 0; and fl rounds to the nearest such number. (The range of e is what (FP3) lets us stop thinking about.) What is u , and what is $\text{fl}(99999996)$?

Solution. Two consecutive representable numbers of the same exponent e differ by 10^{e-5} , and a number t with that exponent has $|t| \geq 10^e$. Rounding to the nearest moves t by at most half the gap, so

$$|\text{fl}(t) - t| \leq \frac{1}{2} \cdot 10^{e-5} \leq \frac{1}{2} \cdot 10^{-5}|t|,$$

which is (FP1) with $u = \frac{1}{2} \cdot 10^{-5} = 5 \times 10^{-6}$.

For $\text{fl}(99999996)$: the number is 9.9999996×10^7 , and the two neighbors with six digits are 9.99999×10^7 and 1.00000×10^8 . The second is nearer, so $\text{fl}(99999996) = 10^8$. Note what happened: eight digits went in, six came out, and the two digits that were lost were the only two that distinguished 99999996 from 10^8 .

Warning. A model with an absolute error bound would be a different, and false, model. Suppose we had written $\text{fl}(x + y) = (x + y) + \delta$ with $|\delta| \leq u$. Then $\text{fl}(10^6 + 1)$ would have to be within $u = 5 \times 10^{-6}$ of $10^6 + 1$, whereas a six-digit machine returns 10^6 , an absolute error of 1. The error a machine makes is proportional to the size of the number it is

storing; a machine is accurate in *ratio*, never in *amount*. Confusing the two is the commonest mistake in the subject, and (FP1) exists to prevent it.

Quick check. In the six-digit model, is $\text{fl}(x+y) = x+y$ possible when x and y are representable and both nonzero? Is $\text{fl}(x-y) = x-y$ possible when $x \neq y$?

Solution. Yes to both, and easily. $1 + 2 = 3$ is representable, so the first addition is exact. For the second, $10^8 - 99999.9 = 99900000.1$, which is *not* representable, so that particular subtraction is inexact; but $1.00001 - 1.00000 = 10^{-5}$ is representable exactly, so that subtraction is exact. The model never claims an operation *is* inexact. It claims only a bound on how inexact it can be. That both extremes occur is what makes the subject interesting: the second subtraction is performed exactly, and it is nevertheless the shape of subtraction that destroys answers, as the last example of this section will show.

24.1.3 Accumulated rounding: the constant γ_k

A single operation costs a factor $1 + \delta$. An algorithm performs many, and the factors pile up: an answer typically carries a product like $(1 + \delta_1)(1 + \delta_2) \cdots (1 + \delta_k)$, sometimes with an inverted factor where a division occurred. Carrying such products around is unbearable, so we bound them once and for all. The constant that results is used in every remaining section of this chapter.

Lemma 24.1.1. Accumulated rounding

Let $k \geq 1$ and let $\delta_1, \dots, \delta_k$ satisfy $|\delta_i| \leq u$, and let $\rho_1, \dots, \rho_k \in \{+1, -1\}$. If $ku < 1$, then

$$\prod_{i=1}^k (1 + \delta_i)^{\rho_i} = 1 + \theta \quad \text{with} \quad |\theta| \leq \gamma_k := \frac{ku}{1 - ku}.$$

Idea. Induction on k , with the two signs treated as two cases. The only thing to watch is that the recursion for γ_k survives division, where the denominator $1 - u$ costs something; the definition $\gamma_k = ku/(1 - ku)$ is exactly the bound that absorbs that cost.

Proof. Note first that $ku < 1$ forces $ju < 1$ for $j \leq k$, so every γ_j with $j \leq k$ is defined and positive, and $\gamma_1 \leq \gamma_2 \leq \cdots \leq \gamma_k$.

Base case $k = 1$. If $\rho_1 = +1$, then $\theta = \delta_1$ and $|\theta| \leq u \leq u/(1 - u) = \gamma_1$. If $\rho_1 = -1$, then

$$\theta = \frac{1}{1 + \delta_1} - 1 = \frac{-\delta_1}{1 + \delta_1}, \quad |\theta| \leq \frac{u}{1 - u} = \gamma_1,$$

where $1 + \delta_1 \geq 1 - u > 0$ because $u < 1$.

Inductive step. Suppose the product of the first $k - 1$ factors is $1 + \theta'$ with $|\theta'| \leq \gamma_{k-1}$, and write $\delta = \delta_k$.

If $\rho_k = +1$, then $1 + \theta = (1 + \theta')(1 + \delta)$, so $\theta = \theta' + \delta + \theta'\delta$ and

$$\begin{aligned} |\theta| &\leq \gamma_{k-1} + u + \gamma_{k-1}u \\ &= \frac{(k-1)u + u(1 - (k-1)u) + (k-1)u^2}{1 - (k-1)u} = \frac{ku}{1 - (k-1)u}, \end{aligned}$$

and $1 - (k-1)u \geq 1 - ku > 0$, so this is at most $ku/(1 - ku) = \gamma_k$.

If $\rho_k = -1$, then $1 + \theta = (1 + \theta')/(1 + \delta)$, so $\theta = (\theta' - \delta)/(1 + \delta)$ and $|\theta| \leq (\gamma_{k-1} + u)/(1 - u)$. It remains to check that this is at most γ_k . Clearing the positive denominators $1 - (k-1)u$, $1 - u$, and $1 - ku$, the inequality $(\gamma_{k-1} + u)/(1 - u) \leq \gamma_k$ is equivalent to

$$u(k - (k-1)u)(1 - ku) \leq ku(1 - ku + (k-1)u^2),$$

and expanding both sides gives $k - k^2u - (k-1)u + k(k-1)u^2$ on the left against $k - k^2u + k(k-1)u^2$ on the right, after dividing by $u > 0$. The left side is the right side minus $(k-1)u$, which is at least 0, so the left side is at most the right side and the inequality holds. This proves the lemma. ■

Remark. Read γ_k as “about ku ”. Indeed if $ku \leq 0.01$ then $\gamma_k \leq ku/0.99 < 1.02ku$. The denominator is there only so that the induction above closes; it is never the point of a bound. We will also use, without further comment, that $\gamma_j \leq \gamma_k$ for $j \leq k$, and we set $\gamma_0 = 0$, which is what the formula gives and what the empty product deserves.

24.1.4 The first backward error analysis

Here is the simplest algorithm in the book: add up n products. What comes out is not $\mathbf{x}^\top \mathbf{y}$. But it is *exactly* $\mathbf{x}^\top \mathbf{y}$ for a slightly different $\mathbf{x}$ — and that turns out to be the useful thing to say.

Theorem 24.1.2. Backward Error of an Inner Product

Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ have representable entries, let $nu < 1$, and let the inner product be computed in the model of [Definition 24.1.1](#) in the natural order,

$$s_1 = \text{fl}(x_1 y_1), \quad s_k = \text{fl}(s_{k-1} + \text{fl}(x_k y_k)) \quad (k = 2, \dots, n).$$

Then there is a vector $\Delta \mathbf{x} \in \mathbb{R}^n$ with

$$s_n = (\mathbf{x} + \Delta \mathbf{x})^\top \mathbf{y} \quad \text{and} \quad |\Delta x_i| \leq \gamma_n |x_i| \quad \text{for every } i.$$

Idea. Unroll the recursion and see which rounding factors touch which term. The term $x_k y_k$ picks up one factor from its own multiplication and one from each addition it takes part in afterwards, so no term collects more than n factors. [Lemma 24.1.1](#) converts “at most n factors” into “ $1 + \theta$ with $|\theta| \leq \gamma_n$ ”, and a factor attached to $x_k y_k$ can be blamed on x_k alone.

Proof. By (FP2) there are numbers δ_k and ε_k , all of modulus at most u , with

$$\begin{aligned} \text{fl}(x_k y_k) &= x_k y_k (1 + \delta_k) \quad (1 \leq k \leq n), \\ s_k &= (s_{k-1} + \text{fl}(x_k y_k))(1 + \varepsilon_k) \quad (2 \leq k \leq n). \end{aligned}$$

An induction on m gives

$$s_m = \sum_{k=1}^m x_k y_k (1 + \delta_k) \prod_{j=\max(k,2)}^m (1 + \varepsilon_j),$$

the empty product being 1: for $m = 1$ this is $s_1 = x_1 y_1 (1 + \delta_1)$, and the step from $m - 1$ to m multiplies every existing term by $1 + \varepsilon_m$ and adds the new term $x_m y_m (1 + \delta_m)(1 + \varepsilon_m)$, which is what the formula says.

Take $m = n$ and count factors in the k -th term. For $k = 1$ there are $1 + (n - 1) = n$ of them; for $k \geq 2$ there are $1 + (n - k + 1) = n - k + 2 \leq n$. Every factor has the form $1 + \delta$ with $|\delta| \leq u$, and $nu < 1$, so [Lemma 24.1.1](#) (with all $\rho_i = +1$) gives numbers θ_k with

$$s_n = \sum_{k=1}^n x_k y_k (1 + \theta_k), \quad |\theta_k| \leq \gamma_n.$$

Put $\Delta x_k = x_k \theta_k$. Then $|\Delta x_k| \leq \gamma_n |x_k|$ and $s_n = \sum_k (x_k + \Delta x_k) y_k = (\mathbf{x} + \Delta \mathbf{x})^\top \mathbf{y}$. This proves the theorem. ■

The conclusion is worth staring at. The computed number is not close to $\mathbf{x}^\top \mathbf{y}$ — if $\mathbf{x}^\top \mathbf{y}$ happens to be tiny compared with $\sum_k |x_k y_k|$, it can be wrong in every digit. What the theorem says is different and stronger in kind: *the computed number is an inner product, exactly, of data within a relative γ_n of the given data*. The algorithm has not solved a nearby problem approximately; it has solved a nearby problem exactly.

Remark. The same proof, with the roles of $\mathbf{x}$ and $\mathbf{y}$ exchanged, gives $s_n = \mathbf{x}^\top (\mathbf{y} + \Delta \mathbf{y})$ with $|\Delta \mathbf{y}| \leq \gamma_n |\mathbf{y}|$ entrywise. What it does **not** give is a perturbation of both at once with half the constant, and it does not give a bound on $|s_n - \mathbf{x}^\top \mathbf{y}|$ relative to $|\mathbf{x}^\top \mathbf{y}|$. [Exercise \(C1: When backward stability says nothing\)](#) makes the last point precise. For complex $\mathbf{x}, \mathbf{y}$ the operations $+$, $\times$ are themselves built from several real operations, so the model must be applied to those; the resulting bound has the same shape with a modestly larger constant, and we do not prove it here or use it.

24.1.5 Forward error, backward error, stability

We can now name what [Theorem 24.1.2](#) did, and the alternative it declined to do. Fix a **problem**: a map $f: D \rightarrow \mathbb{R}^m$ from data to answers, where D is a set of admissible data in some $\mathbb{R}^N$. The set D always consists of **real** data and is never cut down to the representable ones: for the problems of this chapter it is all of $\mathbb{R}^N$, or all invertible matrices, and it has to be, because the perturbed datum $\mathbf{a} + \Delta \mathbf{a}$ below must lie in it. An **algorithm** for f is a map $\hat{f}$ — defined at the representable data, and sending each of them to whatever the computation actually returns in the model, rounding and all. Two numbers measure how the second differs from the first.

Definition 24.1.2. Forward and backward error

Let f be a problem, $\hat{f}$ an algorithm for it, and $\mathbf{a} \in D$ a representable datum with $f(\mathbf{a}) \neq \mathbf{0}$. The **forward error** of $\hat{f}$ at $\mathbf{a}$ is the relative error in the answer,

$$\frac{\|\hat{f}(\mathbf{a}) - f(\mathbf{a})\|}{\|f(\mathbf{a})\|}.$$

The **backward error** of $\hat{f}$ at $\mathbf{a}$ is the smallest relative perturbation of the data that accounts for the computed answer exactly,

$$\inf \left\{ \frac{\|\Delta \mathbf{a}\|}{\|\mathbf{a}\|} : \mathbf{a} + \Delta \mathbf{a} \in D \text{ and } f(\mathbf{a} + \Delta \mathbf{a}) = \hat{f}(\mathbf{a}) \right\},$$

with the convention that it is ∞ when no such $\Delta \mathbf{a}$ exists.

The forward error is what a user wants to know and can almost never compute, since it needs $f(\mathbf{a})$. The backward error is what an analyst can bound, because bounding it only requires *exhibiting* one $\Delta \mathbf{a}$, which a proof like the one above hands over for free. An algorithm whose backward error is always as small as the arithmetic allows is the best one can hope for.

A backward stable algorithm returns the exact answer to a question you might as well have asked.

Definition 24.1.3. Backward stability

Let $f: D \rightarrow \mathbb{R}^m$ be a problem. An algorithm $\hat{f}$ for f , running in a floating-point model with unit roundoff u , is **backward stable** if there is a constant c , depending only on the dimensions of the data and **not** on the data itself or on u , such that for every representable $\mathbf{a} \in D$ there exists $\Delta \mathbf{a} \in \mathbb{R}^N$ with

$$\mathbf{a} + \Delta \mathbf{a} \in D, \quad \hat{f}(\mathbf{a}) = f(\mathbf{a} + \Delta \mathbf{a}), \quad \|\Delta \mathbf{a}\| \leq cu \|\mathbf{a}\|.$$

The small words carry the weight. **For every $\mathbf{a}$:** one lucky datum proves nothing. **c independent of the data:** a “constant” allowed to grow with $\|\mathbf{a}\|$ or with κ would make the definition vacuous. **c independent of u :** the bound must improve when the arithmetic does. The constant is allowed to depend on n , and in practice always does — γ_n is about nu , not u — which is why “backward stable” is a statement about a *family* of algorithms indexed by size, and why nobody worries much about a factor of n .

By [Theorem 24.1.2](#), the natural inner product algorithm is backward stable as an algorithm in $\mathbf{x}$ for the problem $\mathbf{x} \mapsto \mathbf{x}^\top \mathbf{y}$ with $\mathbf{y}$ fixed, on the domain $D = \mathbb{R}^n$. Take $c = 1.02n$, which depends on the dimension alone. Indeed the entrywise bound $|\Delta \mathbf{x}| \leq \gamma_n |\mathbf{x}|$ implies $\|\Delta \mathbf{x}\| \leq \gamma_n \|\mathbf{x}\|$ in each of the norms $\|\cdot\|_1$, $\|\cdot\|_2$, $\|\cdot\|_\infty$, and $\gamma_n \leq 1.02nu$ in every model with $nu \leq 0.01$, by the remark after [Lemma 24.1.1](#). (Restricting to such models is how a bound stated with γ_n is turned into one with a constant independent of u , and we do it silently from here on.)

Some algorithms are not backward stable and are not bad either: they perturb the data a little *and* the answer a little, and neither perturbation alone accounts for the result. The weaker notion that covers them is worth naming, so that we never have to claim more than we have proved.

Definition 24.1.4. Mixed forward-backward stability

An algorithm $\hat{f}$ for $f: D \rightarrow \mathbb{R}^m$ is **mixed forward-backward stable** if there is a constant c , depending only on the dimensions, such that for every representable $\mathbf{a} \in D$ there are $\Delta\mathbf{a}$ and $\Delta\mathbf{y}$ with $\mathbf{a} + \Delta\mathbf{a} \in D$ and

$$\hat{f}(\mathbf{a}) = f(\mathbf{a} + \Delta\mathbf{a}) + \Delta\mathbf{y}, \quad \|\Delta\mathbf{a}\| \leq cu\|\mathbf{a}\|, \quad \|\Delta\mathbf{y}\| \leq cu\|f(\mathbf{a} + \Delta\mathbf{a})\|.$$

Backward stability is the special case $\Delta\mathbf{y} = \mathbf{0}$, so it is the stronger statement of the two. A mixed stable algorithm does **not** satisfy the hypothesis of the next theorem: its $\Delta\mathbf{y}$ contributes a further term to the forward error, and we do not estimate that term anywhere in this chapter. Where a method in this chapter is only mixed stable, we say so, and we claim nothing more about it than the definition gives.

Warning. Backward stable does not mean accurate, and inaccurate does not mean unstable. A backward stable algorithm commits to nothing about the forward error except what the *problem* allows, and for an ill-conditioned problem that is very little. Conversely, an algorithm can return an accurate answer on one datum by luck while being backward unstable. The two words describe different objects: conditioning belongs to f , stability belongs to $\hat{f}$. Chapter 16 §08 promised exactly this: “Chapter 24 draws the distinction properly, defines backward stability, and proves which of the algorithms in this book have it.” Definition 24.1.3 is the definition; the proofs occupy Sections 2 and 3, and the next theorem is the only bridge between the two sides.

24.1.6 Multiplying the two together

The rule of thumb everyone quotes is *forward error* $\lesssim$ *condition number* $\times$ *backward error*. For linear systems it is not a rule of thumb at all. It is Theorem 16.8.3 with a different reading of the same symbols.

Theorem 24.1.3. Forward Error from Backward Error

Let $\mathbf{A} \in M_n(\mathbb{C})$ be invertible, $\mathbf{b} \neq \mathbf{0}$, and $\mathbf{Ax} = \mathbf{b}$. Let $\hat{\mathbf{x}}$ be any vector which is the exact solution of a perturbed system,

$$(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b}, \quad \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \leq \eta,$$

and suppose $r := \kappa(\mathbf{A})\eta < 1$. Then $\mathbf{A} + \Delta\mathbf{A}$ is invertible and

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{r}{1-r} = \frac{\kappa(\mathbf{A})\eta}{1 - \kappa(\mathbf{A})\eta}.$$

In particular, if $\eta = cu$ is admissible — as it is when the algorithm producing $\hat{\mathbf{x}}$ is backward stable with constant c **for the problem** $\mathbf{A} \mapsto \mathbf{A}^{-1}\mathbf{b}$ with $\mathbf{b}$ held fixed, so that only $\mathbf{A}$ is perturbed — and if $\kappa(\mathbf{A})cu \leq \frac{1}{2}$, then the forward relative error is at most $2\kappa(\mathbf{A})cu$.

Idea. There is nothing to do: backward stability says the computed answer solves a perturbed system exactly, and Chapter 16 already bounded the effect of perturbing $\mathbf{A}$. The content of

the theorem is the *translation*, not the estimate.

Proof. Since $\|\Delta\mathbf{A}\|/\|\mathbf{A}\| \leq \eta$, we have $\kappa(\mathbf{A})\|\Delta\mathbf{A}\|/\|\mathbf{A}\| \leq r < 1$, so Theorem 16.8.3 (b), applied with $\mathbf{E} = \Delta\mathbf{A}$, says that $\mathbf{A} + \Delta\mathbf{A}$ is invertible and that

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{r'}{1 - r'}, \quad r' := \kappa(\mathbf{A}) \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \leq r.$$

The function $s \mapsto s/(1 - s)$ is increasing on $[0, 1)$, since its derivative $(1 - s)^{-2}$ is positive there, so $r'/(1 - r') \leq r/(1 - r)$. For the last claim put $\eta = cu$; then $r = \kappa(\mathbf{A})cu \leq \frac{1}{2}$ gives $1 - r \geq \frac{1}{2}$ and $r/(1 - r) \leq 2r$. This proves the theorem. ■

Remark. **When the right-hand side is perturbed too.** A solver for $\mathbf{Ax} = \mathbf{b}$ normally perturbs the pair $(\mathbf{A}, \mathbf{b})$, returning $\hat{\mathbf{x}} \neq \mathbf{0}$ with

$$(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b} + \Delta\mathbf{b}, \quad \|\Delta\mathbf{A}\|_2 \leq cu\|\mathbf{A}\|_2, \quad \|\Delta\mathbf{b}\|_2 \leq cu\|\mathbf{b}\|_2,$$

which is not literally the hypothesis above. One absorption repairs that. Put $\Delta\mathbf{A}' = \Delta\mathbf{A} - \Delta\mathbf{b}\hat{\mathbf{x}}^*/\|\hat{\mathbf{x}}\|_2^2$; then $(\mathbf{A} + \Delta\mathbf{A}')\hat{\mathbf{x}} = \mathbf{b} + \Delta\mathbf{b} - \Delta\mathbf{b} = \mathbf{b}$, and the subtracted matrix is an outer product, of 2-norm $\|\Delta\mathbf{b}\|_2/\|\hat{\mathbf{x}}\|_2$. So Theorem 24.1.3 applies in the 2-norm with

$$\eta = cu \left(1 + \frac{\|\mathbf{b}\|_2}{\|\mathbf{A}\|_2\|\hat{\mathbf{x}}\|_2} \right) \leq cu \left(1 + \frac{\|\mathbf{x}\|_2}{\|\hat{\mathbf{x}}\|_2} \right),$$

the last step because $\|\mathbf{b}\|_2 = \|\mathbf{Ax}\|_2 \leq \|\mathbf{A}\|_2\|\mathbf{x}\|_2$. That is a small multiple of cu as soon as the computed $\hat{\mathbf{x}}$ is not much shorter than $\mathbf{x}$, and it is how every later appeal to the theorem for a linear solver is to be read.

Reading the bound in digits. Suppose the arithmetic has $u \approx 10^{-16}$ and the algorithm is backward stable with a modest c . The bound then reads: relative forward error at most about $\kappa(\mathbf{A}) \cdot 10^{-16}$. Taking logarithms, the computed answer may be wrong from about the $(16 - \log_{10} \kappa(\mathbf{A}))$ -th significant digit onward. This is the origin of the sentence “a stable solve loses about $\log_{10} \kappa$ decimal digits”. Two cautions, both real. It is an upper bound, so the loss may be smaller — Chapter 16 §08’s warning that “the condition number bounds the error; it does not produce it” applies verbatim. And it is a statement about $\kappa(\mathbf{A})$ for the *problem as posed*: an algorithm that quietly poses a different problem pays that problem’s condition number instead.

That last point is the whole of Chapter 16 §08’s remaining ledger. Its third unpaid item read: “**Hence the comparison of the two algorithms** — that a stable implementation of the QR route loses about $\log_{10} \kappa_2(\mathbf{A})$ decimal digits while forming and solving the normal equations loses about $2 \log_{10} \kappa_2(\mathbf{A})$ — is still Chapter 24’s to prove.” The theorem above supplies the first half of the sentence for any backward stable method. Section 3 supplies the second half and the two algorithms, using Chapter 16’s own Theorem 16.8.4 for the square.

Quick check. An algorithm for $\mathbf{Ax} = \mathbf{b}$ is backward stable, $\kappa(\mathbf{A}) = 10^8$, and $u = 10^{-16}$. A user reports that the computed $\hat{\mathbf{x}}$ has only eight correct digits and concludes that the

algorithm is broken. What is wrong with the conclusion?

Solution. Nothing at all is wrong with the algorithm: [Theorem 24.1.3](#) — applied after absorbing any perturbation of $\mathbf{b}$ into $\Delta\mathbf{A}$, as in the remark following it — predicts a relative forward error of about $\kappa(\mathbf{A})u = 10^{-8}$, up to the modest constant c , which is exactly eight correct digits. The loss is charged to the problem, not to the method. Chapter 16 §08 put it as sharply as it can be put: with data of limited accuracy, an ill-conditioned answer *is not determined* to more digits than that, whatever anybody computes with.

24.1.7 Cancellation, and where the damage really happens

Backward stability is a property of a whole algorithm, and algorithms are usually wrecked in one place. That place is almost always a subtraction of two nearly equal numbers. The following example is the standard one, and it repays being done digit by digit.

Example. Two ways to solve a quadratic

In the six-digit decimal model of [Example \(The model we compute in\)](#), find the two roots of

$$x^2 - 10^4x + 1 = 0$$

by the quadratic formula $x_{\pm} = \frac{1}{2}(10^4 \pm \sqrt{10^8 - 4})$, and then again with the smaller root computed as $x_- = 1/x_+$. Compare with the exact roots.

Solution. The exact roots. Their sum is 10^4 and their product is 1. Now $9999.9998^2 = 10^8 - 4 + 4 \times 10^{-8}$, which is just above $10^8 - 4$, so $\sqrt{10^8 - 4} = 9999.99979999\dots$ and

$$x_+ = 9999.99989999\dots, \quad x_- = \frac{1}{x_+} = 1.0000000100\dots \times 10^{-4}.$$

The formula, step by step. Each line is one operation of the model.

- $\text{fl}(10^4 \times 10^4) = 10^8$, exactly representable, no error.
- $\text{fl}(10^8 - 4) = \text{fl}(99999996) = 1.00000 \times 10^8$, by [Example \(The model we compute in\)](#). **Here the two informative digits are lost**, and the relative error committed is about 4×10^{-8} , comfortably inside $u = 5 \times 10^{-6}$. The model is behaving perfectly.
- $\text{fl}(\sqrt{10^8}) = 10^4$, exactly.
- $\text{fl}(10^4 + 10^4) = 2 \times 10^4$ and $\text{fl}(2 \times 10^4 \div 2) = 10^4$: the computed x_+ is 10^4 , with relative error 10^{-8} . Excellent.
- $\text{fl}(10^4 - 10^4) = 0$, exactly, and $\text{fl}(0 \div 2) = 0$: the computed x_- is **0**.

The repair. Compute $x_+ = 10^4$ as above, then $\text{fl}(1 \div 10^4) = 10^{-4}$, whose relative error against $1.00000001 \times 10^{-4}$ is 10^{-8} .

What to conclude. The second route loses nothing; the first returns a root with no correct digit, indeed with no digits at all. And notice **where** the damage was done. The fatal subtraction $10^4 - 10^4$ was performed **exactly**: both operands were representable and so

was their difference. The error was already present, manufactured four lines earlier when 99999996 was rounded to 10^8 , and the subtraction merely deleted the digits the two operands had in common, which was all of them. Cancellation does not create error. It *exposes* error, by deleting the digits that were correct and promoting the ones that were not.

Warning. A small residual is not a small error, and a small error is not a small residual. For a computed solution $\hat{\mathbf{x}}$ of $\mathbf{Ax} = \mathbf{b}$ with $\hat{\mathbf{x}} \neq \mathbf{0}$, put $\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$ and

$$\Delta\mathbf{A} = \frac{\mathbf{r}\hat{\mathbf{x}}^*}{\|\hat{\mathbf{x}}\|_2^2}, \quad \text{so that} \quad (\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{r} = \mathbf{b},$$

and $\|\Delta\mathbf{A}\|_2 = \|\mathbf{r}\|_2/\|\hat{\mathbf{x}}\|_2$: Cauchy–Schwarz gives $\leq$, and $\Delta\mathbf{A}\hat{\mathbf{x}} = \mathbf{r}$ attains it — the computation is the one in the proof of Theorem 20.4.4, with $\mathbf{b}$ in place of $\mu\hat{\mathbf{x}}$. So a small relative residual *is* a small backward error, and nothing more: by Theorem 24.1.3 the forward error may still be $\kappa(\mathbf{A})$ times as large. In the other direction, an accurate $\hat{\mathbf{x}}$ for a badly scaled $\mathbf{A}$ can have a residual far larger than one expects. Chapter 20 §04’s Theorem 20.4.4 is the eigenvalue version of exactly this move, and that section named the technique: “An error analysis of this kind, which measures an answer by a change in the data that makes it exact, is a **backward error analysis**. Chapter 24 builds on it.”

24.1.8 Exercises

24.1.8.1 A. Check your understanding

Exercise. A1

- State the three clauses of Definition 24.1.1, and say which one is an exclusion rather than an assumption about accuracy.
- Define the unit roundoff, and give its value for a decimal model carrying t significant digits.
- Define the forward error and the backward error of an algorithm at a datum, and say which of the two a user can usually measure.
- Decide whether each statement is correct, with a reason: (i) “a backward stable algorithm returns an answer with small forward error”; (ii) “if the computed $\hat{\mathbf{x}}$ has a tiny residual, then $\hat{\mathbf{x}}$ is close to $\mathbf{x}$ ”.
- Why is Definition 24.1.1 called a model rather than a theorem?

Solution.

- (FP1) rounding is relative, with error at most u ; (FP2) each operation returns the rounding of its exact result, so $\text{fl}(x \odot y) = (x \odot y)(1 + \delta)$ with $|\delta| \leq u$; (FP3) no overflow or underflow. (FP3) is the exclusion: it does not say the arithmetic is good, it says we decline to analyze the case where it is not.
- u is the bound in (FP1): $|\text{fl}(t) - t| \leq u|t|$ for every in-range t . For a decimal model with t significant digits, $u = \frac{1}{2} \cdot 10^{1-t}$ (Example (The model we compute in) does the case $t = 6$).

- (c) See [Definition 24.1.2](#). The forward error is $\|\hat{f}(\mathbf{a}) - f(\mathbf{a})\|/\|f(\mathbf{a})\|$, the backward error the smallest $\|\Delta\mathbf{a}\|/\|\mathbf{a}\|$ with $f(\mathbf{a} + \Delta\mathbf{a}) = \hat{f}(\mathbf{a})$. A user can usually measure neither directly, but the backward error is the one an analyst can *bound*, because a bound needs only one exhibited $\Delta\mathbf{a}$; computing the forward error needs the exact answer, which is what the computation was for.
- (d) (i) Incorrect. It returns an answer with small *backward* error; the forward error is then at most about κ times that, by [Theorem 24.1.3](#) (after absorbing any perturbation of $\mathbf{b}$, as the remark after that theorem does), and for large κ that is no guarantee at all. (ii) Incorrect, for the same reason: a small residual is a small backward error (see the warning above), which bounds the forward error only after multiplication by $\kappa(\mathbf{A})$.
- (e) Because it is an assumption about a physical machine, not a consequence of anything proved earlier in the book. Nothing in Chapters 1 to 21 mentions rounding. Everything this chapter proves about arithmetic is a theorem *about the model*, true of any machine that satisfies it and silent about one that does not.

24.1.8.2 B. Practice

Exercise. B1: The γ calculus

Assume $(j + k)u < 1$.

- (a) Prove that $1 + \gamma_k = 1/(1 - ku)$.
- (b) Hence prove that $\gamma_j + \gamma_k + \gamma_j\gamma_k \leq \gamma_{j+k}$.
- (c) Explain in one sentence why (b) is the statement one needs when two separately analyzed computations are performed one after the other.

Solution.

- (a) $1 + \gamma_k = 1 + ku/(1 - ku) = ((1 - ku) + ku)/(1 - ku) = 1/(1 - ku)$.
- (b) Using (a) twice,

$$1 + \gamma_j + \gamma_k + \gamma_j\gamma_k = (1 + \gamma_j)(1 + \gamma_k) = \frac{1}{(1 - ju)(1 - ku)}.$$

Now $(1 - ju)(1 - ku) = 1 - (j + k)u + jku^2 \geq 1 - (j + k)u > 0$, and inverting reverses the inequality between positive numbers, so the right-hand side is at most $1/(1 - (j + k)u) = 1 + \gamma_{j+k}$. Subtracting 1 gives the claim.

- (c) If one stage multiplies the exact answer by $1 + \theta_1$ with $|\theta_1| \leq \gamma_j$ and the next by $1 + \theta_2$ with $|\theta_2| \leq \gamma_k$, the combined factor is $1 + \theta_1 + \theta_2 + \theta_1\theta_2$, and (b) says the total is still of the form $1 + \theta$ with $|\theta| \leq \gamma_{j+k}$ — the counts simply add.

Exercise. B2: Summation

Let $x_1, \dots, x_n$ be representable, let $nu < 1$, and let s_n be computed by $s_1 = x_1$, $s_k = \text{fl}(s_{k-1} + x_k)$.

- (a) Prove that $s_n = \sum_{k=1}^n x_k(1 + \theta_k)$ with $|\theta_k| \leq \gamma_{n-1}$.

- (b) Deduce $|s_n - \sum_k x_k| \leq \gamma_{n-1} \sum_k |x_k|$, and explain why this is **not** a bound on the relative error of s_n .
- (c) Give three representable numbers in the six-digit model for which the computed sum has relative error larger than u .

Solution.

- (a) By (FP2), $s_k = (s_{k-1} + x_k)(1 + \varepsilon_k)$ for $2 \leq k \leq n$, with $|\varepsilon_k| \leq u$. Unrolling as in the proof of [Theorem 24.1.2](#),

$$s_n = \sum_{k=1}^n x_k \prod_{j=\max(k,2)}^n (1 + \varepsilon_j).$$

The $k = 1$ and $k = 2$ terms carry $n-1$ factors and the rest carry fewer, so [Lemma 24.1.1](#) gives $|\theta_k| \leq \gamma_{n-1}$.

- (b) Subtract $\sum_k x_k$ and use the triangle inequality: $|s_n - \sum_k x_k| = |\sum_k x_k \theta_k| \leq \gamma_{n-1} \sum_k |x_k|$. It is not a relative error bound because the right-hand side involves $\sum_k |x_k|$, not $|\sum_k x_k|$, and the two differ by an arbitrarily large factor when the x_k have mixed signs and cancel.
- (c) Take $x_1 = 1$, $x_2 = 10^{-7}$, $x_3 = -1$, summed in that order. Then $\text{fl}(1 + 10^{-7}) = 1$, since $1 + 10^{-7}$ lies strictly between the representable neighbors 1 and 1.00001 and is nearer the first; and $\text{fl}(1 - 1) = 0$. The computed sum is 0, the true sum is 10^{-7} , and the relative error is 1. This is (b) at work: $\sum |x_k| = 2$ while $|\sum x_k| = 10^{-7}$.

Exercise. B3: The model by hand

Work in the six-digit decimal model.

- (a) Compute $\text{fl}(1/3)$ and verify (FP1) for it.
- (b) Compute the two sides of $\text{fl}(\text{fl}(a + b) + c)$ and $\text{fl}(a + \text{fl}(b + c))$ for $a = 10^5$, $b = 1$, $c = 1$, and say what this shows about the model.

Solution.

- (a) $1/3 = 3.33333\bar{3} \times 10^{-1}$, so $\text{fl}(1/3) = 0.333333$ and $|\text{fl}(1/3) - 1/3| = \frac{1}{3} \times 10^{-6}$. Dividing by $1/3$ gives a relative error of 10^{-6} , which is at most $u = 5 \times 10^{-6}$.
- (b) Left: $\text{fl}(10^5 + 1) = 100001$ exactly (six digits), and $\text{fl}(100001 + 1) = 100002$. Right: $\text{fl}(1 + 1) = 2$, and $\text{fl}(10^5 + 2) = 100002$. Here the two agree. Change a to 10^6 : left gives $\text{fl}(10^6 + 1) = 10^6$, then $\text{fl}(10^6 + 1) = 10^6$; right gives $\text{fl}(1 + 1) = 2$, then $\text{fl}(10^6 + 2) = 10^6$ as well, since 1000002 rounds to 1.00000×10^6 . Try $a = 10^6$, $b = c = 3$: left is 10^6 then 10^6 ; right is 6 then $\text{fl}(1000006) = 1.00001 \times 10^6$. The two differ. **Floating-point addition is commutative but not associative**, so the *order* of a summation is part of the algorithm, not an irrelevance — which is why [Theorem 24.1.2](#) names the order it analyzes.

24.1.8.3 C. Going deeper

Exercise. C1: When backward stability says nothing

Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and let s_n be the computed inner product of [Theorem 24.1.2](#).

- (a) Prove that $|s_n - \mathbf{x}^\top \mathbf{y}| \leq \gamma_n |\mathbf{x}|^\top |\mathbf{y}|$, where $|\cdot|$ is entrywise.
- (b) Deduce that if all $x_i y_i \geq 0$ then the *relative* error of s_n is at most γ_n .
- (c) Give $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$ with representable entries, in the six-digit model, for which the relative error of s_n is at least 1. Explain why this does not contradict [Theorem 24.1.2](#).

Solution.

- (a) By the theorem's proof, $s_n = \sum_k x_k y_k (1 + \theta_k)$ with $|\theta_k| \leq \gamma_n$, so

$$|s_n - \mathbf{x}^\top \mathbf{y}| = \left| \sum_k x_k y_k \theta_k \right| \leq \gamma_n \sum_k |x_k| |y_k| = \gamma_n |\mathbf{x}|^\top |\mathbf{y}|.$$

- (b) If every $x_k y_k \geq 0$ then $|\mathbf{x}|^\top |\mathbf{y}| = \sum_k x_k y_k = \mathbf{x}^\top \mathbf{y}$, so (a) reads $|s_n - \mathbf{x}^\top \mathbf{y}| \leq \gamma_n \mathbf{x}^\top \mathbf{y}$, which is the relative bound (and $\mathbf{x}^\top \mathbf{y} > 0$ unless every term vanishes, in which case $s_n = 0$ too).
- (c) Take $\mathbf{x} = (1, 10^{-7}, -1)$ and $\mathbf{y} = (1, 1, 1)$, all representable. Each product $x_k y_k$ is computed exactly, so $\delta_1 = \delta_2 = \delta_3 = 0$. Then $s_1 = 1$; next $s_2 = \text{fl}(1 + 10^{-7}) = 1$, which is $(1 + 10^{-7})(1 + \varepsilon_2)$ with

$$\varepsilon_2 = \frac{-10^{-7}}{1 + 10^{-7}}, \quad |\varepsilon_2| < 10^{-7} < u;$$

finally $s_3 = \text{fl}(1 - 1) = 0$, exactly, so $\varepsilon_3 = 0$ serves. The true value is $\mathbf{x}^\top \mathbf{y} = 10^{-7}$, so the forward relative error is 1.

There is no contradiction. Following the proof, $\theta_1 = \theta_2 = \varepsilon_2$ and $\theta_3 = 0$, all of modulus below $u \leq \gamma_3$, and indeed

$$(1 + \varepsilon_2) + 10^{-7}(1 + \varepsilon_2) - 1 = (1 + 10^{-7})(1 + \varepsilon_2) - 1 = 0 = s_3.$$

So the backward perturbation $\Delta \mathbf{x} = (\varepsilon_2, 10^{-7} \varepsilon_2, 0)$ is of relative size about 10^{-7} in every coordinate: the algorithm solved a neighboring problem exactly. What made the answer worthless is the *problem*, not the method: here $|\mathbf{x}|^\top |\mathbf{y}| = 2 + 10^{-7}$ while $\mathbf{x}^\top \mathbf{y} = 10^{-7}$, so part (a)'s bound is twenty million times the quantity being computed. This is the same phenomenon as [Example \(Two ways to solve a quadratic\)](#), seen through the backward error.

Exercise. C2: An unstable way to compute a variance

Given $x_1, \dots, x_N$ with mean $\bar{x} = \frac{1}{N} \sum_i x_i$, the quantity $S = \sum_i (x_i - \bar{x})^2$ can be computed in one pass as $S = \sum_i x_i^2 - \frac{1}{N} (\sum_i x_i)^2$, or in two passes by forming $\bar{x}$ first.

- (a) Verify the identity $\sum_i (x_i - \bar{x})^2 = \sum_i x_i^2 - \frac{1}{N} (\sum_i x_i)^2$.
- (b) In the six-digit model, run both routes on $x = (10000, 10001, 10002)$, showing every operation, and compare with the exact value $S = 2$.
- (c) Say which route is to be preferred and why, in the vocabulary of this section.

Solution.

- (a) Expand: $\sum_i (x_i - \bar{x})^2 = \sum_i x_i^2 - 2\bar{x} \sum_i x_i + N\bar{x}^2$. Since $\sum_i x_i = N\bar{x}$, the last two terms are $-2N\bar{x}^2 + N\bar{x}^2 = -N\bar{x}^2 = -\frac{1}{N}(\sum_i x_i)^2$.
- (b) *One pass.* The squares are 10^8 , 100020001, 100040004; rounding each to six digits gives 1.00000×10^8 , 1.00020×10^8 , and 1.00040×10^8 , and accumulating gives $\text{fl}(1.00000 \times 10^8 + 1.00020 \times 10^8) = 2.00020 \times 10^8$ and then 3.00060×10^8 . The sum $\sum_i x_i = 30003$ is exact, its square is 900180009, which rounds to 9.00180×10^8 , and dividing by 3 gives 3.00060×10^8 . Subtracting: $\text{fl}(3.00060 \times 10^8 - 3.00060 \times 10^8) = \mathbf{0}$.
Two passes. $\bar{x} = \text{fl}(30003/3) = 10001$, exactly. The deviations $-1, 0, 1$ are computed exactly, their squares are 1, 0, 1, and the sum is **2**, the exact answer.
- (c) The two-pass route. The one-pass route subtracts two nine-digit numbers that agree in their first six digits, so in a six-digit model it commits catastrophic cancellation and returns 0; worse, 0 is not the value of S for *any* small relative perturbation of the data. Indeed $S(\mathbf{x} + \Delta\mathbf{x}) = 0$ forces the three perturbed entries to be equal, say to v ; and for $\mathbf{x} = (10000, 10001, 10002)$ the largest of $|x_i - v|$ is at least 1, whatever v is, because $x_3 - x_1 = 2$. So every $\Delta\mathbf{x}$ that accounts for the computed answer has

$$\frac{\|\Delta\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} \geq \frac{1}{10002} > 19u,$$

whereas [Definition 24.1.3](#) allows *cu*. One datum bounds the constant rather than disposing of it — what this shows exactly is that no $c \leq 19$ will serve for the one-pass route — but the mechanism is general, and it is the one [Example \(Two ways to solve a quadratic\)](#) identified. The two-pass route performs the subtraction $x_i - \bar{x}$ *first*, while the operands are still exactly representable, and nothing is ever lost. Same identity, same arithmetic, different algorithm — which is precisely the distinction between a problem and a method.