Notes on Numerical Stability

Robert A. van de Geijn
The University of Texas
Austin, TX 78712

October 10, 2014

Based on
"Goal-Oriented and Modular Stability Analysis" [3, 4]
by
Paolo Bientinesi and Robert van de Geijn

NOTE: I have not thoroughly proof-read these notes!!!

1 Motivation

Correctness in the presence of error (e.g., when floating point computations are performed) takes on a different meaning. For many problems for which computers are used, there is one correct answer and we expect that answer to be computed by our program. The problem is that most real numbers cannot be stored exactly in a computer memory. They are stored as approximations, floating point numbers, instead. Hence storing them and/or computing with them inherently incurs error. The question thus becomes "When is a program correct in the presense of such errors?"

Let us assume that we wish to evaluate the mapping $f : \mathcal{D} \rightarrow \mathcal{R}$ where $\mathcal{D} \subset \mathbb{R}^n$ is the domain and $\mathcal{R} \subset \mathbb{R}^m$ is the range (codomain). Now, we will let $\hat{f} : \mathcal{D} \rightarrow \mathcal{R}$ denote a computer implementation of this function. Generally, for $x \in \mathcal{D}$ it is the case that $f(x) \neq \hat{f}(x)$. Thus, the computed value is not "correct". From the Notes on Conditioning, we know that it may not be the case that $\hat{f}(x)$ is "close to" $f(x)$. After all, even if $\hat{f}$ is an exact implementation of f, the mere act of storing x may introduce a small error δx and $f(x + \delta x)$ may be far from $f(x)$ if f is ill-conditioned.

The following defines a property that captures correctness in the presense of the kinds of errors that are introduced by computer arithmetic:

Definition 1 *Let given the mapping $f : D \rightarrow R$, where $D \subset \mathbb{R}^n$ is the domain and $R \subset \mathbb{R}^m$ is the range (codomain), let $\hat{f} : D \rightarrow R$ be a computer implementation of this function. We will call $\hat{f}$ a (numerically)* **stable** *implementation of f on domain $\mathcal{D}$ if for all $x \in D$ there exists a $\hat{x}$ "close" to x such that $\hat{f}(x) = f(\hat{x})$.*

In other words, $\hat{f}$ is a stable implementation if the error that is introduced is similar to that introduced when f is evaluated with a slightly changed input. This is illustrated in Figure 1 for a specific input x. If an implemention is not stable, it is numerically unstable.

2 Floating Point Numbers

Only a finite number of (binary) digits can be used to store a real number number. For so-called single-precision and double-precision floating point numbers 32 bits and 64 bits are typically employed, respectively. Let us focus on double precision numbers.

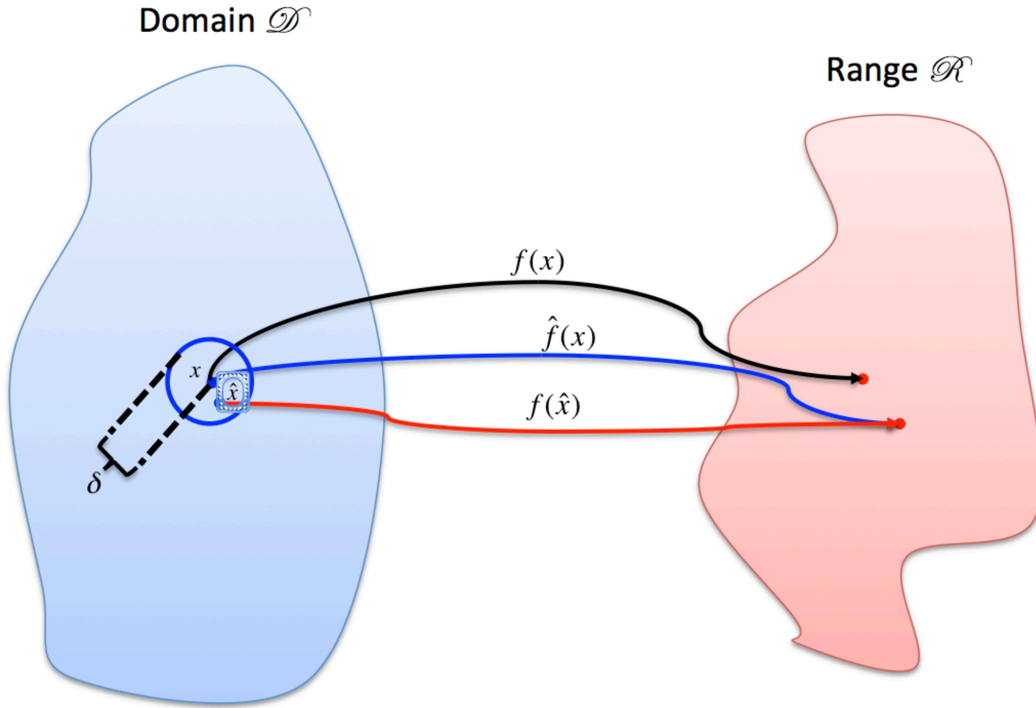

Figure 1: In this illustation, $f : \mathcal{D} \rightarrow \mathcal{R}$ is a function to be evaluated. The function $\check{f}$ represents the implementation of the function that uses floating point arithmetic, thus incurring errors. The fact that for a nearby value $\check{x}$ the computed value equals the exact function applied to the slightly perturbed x, $f(\check{x}) = \check{f}(x)$, means that the error in the computation can be attributed to a small change in the input. If this is true, then $\check{f}$ is said to be a (numerically) stable implementation of f for input x.

Recall that any real number can be written as $\mu \times \beta^e$, where β is the base (an integer greater than one), $\mu \in (-1, 1)$ is the mantissa, and e is the exponent (an integer). For our discussion, we will define F as the set of all numbers $\chi = \mu\beta^e$ such that $\beta = 2$, $\mu = \pm.\delta_0\delta_1 \cdots \delta_{t-1}$ has only t (binary) digits ($\delta_j \in \{0, 1\}$), $\delta_0 = 0$ iff $\mu = 0$ (the mantissa is normalized), and $-L \leq e \leq U$. Elements in F can be stored with a finite number of (binary) digits.

- There is a largest number (in absolute value) that can be stored. Any number that is larger "overflows". Typically, this causes a value that denotes a NaN (Not-a-Number) to be stored.
- There is a smallest number (in absolute value) that can be stored. Any number that is smaller "underflows". Typically, this causes a zero to be stored.

Example 2 *For $x \in \mathbb{R}^n$, consider computing*

$$\|x\|_2 = \sqrt{\sum_{i=0}^{n-1} \chi_i^2}. \tag{1}$$

Notice that

$$\|x\|_2 \leq \sqrt{n} \max_{i=0}^{n-1} |\chi_i|$$

and hence unless some χ_i is close to overflowing, the result will not overflow. The problem is that if some element χ_i has the property that χ_i^2 overflows, intermediate results in the computation in (1) will overflow. The solution is to determine k such that

$$|\chi_k| = max_{i=0}^{n-1}|\chi_i|$$

and to then instead compute

$$\|x\|_2 = |\chi_k|\sqrt{\sum_{i=0}^{n-1}\left(\frac{\chi_i}{\chi_k}\right)^2}.$$

It can be argued that the same approach also avoids underflow if underflow can be avoided..

In our further discussions, we will ignore overflow and underflow issues.

What is important is that any time a real number is stored in our computer, it is stored as the nearest floating point number (element in F). We first assume that it is truncated (which makes our explanation slightly simpler).

Let positive χ be represented by

$$\chi = .\delta_0\delta_1 \cdots \times 2^e,$$

where $\delta_0 = 1$ (the mantissa is normalized). If t digits are stored by our floating point system, then $\hat{\chi} = .\delta_0\delta_1 \cdots \delta_{t-1} \times 2^e$ is stored. Let $\delta\chi = \chi - \hat{\chi}$. Then

$$\delta\chi = \underbrace{.\delta_0\delta_1 \cdots \delta_{t-1}\delta_t \cdots \times 2^e}_{\chi} - \underbrace{.\delta_0\delta_1 \cdots \delta_{t-1} \times 2^e}_{\hat{\chi}} = .\underbrace{0 \cdots 0}_{t}\delta_t \cdots \times 2^e < .\underbrace{0 \cdots 01}_{t} \times 2^e = 2^{e-t}.$$

Also, since χ is positive,

$$\chi = .\delta_0\delta_1 \cdots \times 2^e \geq .1 \times 2^e \geq 2^{e-1}.$$

Thus,

$$\frac{\delta\chi}{\chi} \leq \frac{2^{e-t}}{2^{e-1}} = 2^{1-t}$$

which can also be written as

$$\delta\chi \leq 2^{1-t}\chi.$$

A careful analysis of what happens when χ might equal zero or be negative yields

$$|\delta\chi| \leq 2^{1-t}|\chi|.$$

Now, in practice any base β can be used and floating point computation uses rounding rather than truncating. A similar analysis can be used to show that then

$$|\delta\chi| \leq \mathbf{u}|\chi|$$

where $\mathbf{u} = \frac{1}{2}\beta^{1-t}$ is known as the **machine epsilon** or **unit roundoff**. When using single precision or double precision real arithmetic, $\mathbf{u} \approx 10^{-8}$ or 10^{-16}, respectively. The quantity $\mathbf{u}$ is machine dependent; it is a function of the parameters characterizing the machine arithmetic. The unit roundoff is often alternatively defined as the maximum positive floating point number which can be added to the number stored as 1 without changing the number stored as 1. In the notation introduced below, $[1 + \mathbf{u}] = 1$.

Exercise 3 *Assume a floating point number system with $\beta = 2$ and a mantissa with t digits so that a typical positive number is written as $.d_0d_1 \ldots d_{t-1} \times 2^e$, with $d_i \in \{0, 1\}$.*

- *Write the number* 1 *as a floating point number.*

  Answer: $.\underbrace{10\cdots0}_{\substack{t\\ \text{digits}}} \times 2^1$.

  End of Answer:

- *What is the largest positive real number* $\mathbf{u}$ *(represented as a binary fraction) such that the floating point representation of* $1+\mathbf{u}$ *equals the floating point representation of* 1*? (Assume rounded arithmetic.)*

- *Show that* $\mathbf{u} = \frac{1}{2}2^{1-t}$.

3 Notation

When discussing error analyses, we will distinguish between exact and computed quantities. The function $[expression]$ returns the result of the evaluation of *expression*, where every operation is executed in floating point arithmetic. For example, assuming that the expressions are evaluated from left to right, $[\chi + \psi + \zeta/\omega]$ is equivalent to $[[[\chi] + [\psi]] + [[\zeta] \,/\, [\omega]]]$. Equality between the quantities *lhs* and *rhs* is denoted by $lhs = rhs$. Assignment of *rhs* to *lhs* is denoted by $lhs := rhs$ (*lhs* becomes *rhs*). In the context of a program, the statements $lhs := rhs$ and $lhs := [rhs]$ are equivalent. Given an assignment $\kappa := expression$, we use the notation $\check{\kappa}$ (pronounced "check kappa") to denote the quantity resulting from $[expression]$, which is actually stored in the variable κ.

4 Floating Point Computation

We introduce definitions and results regarding floating point arithmetic. In this note, we focus on real valued arithmetic only. Extensions to complex arithmetic are straightforward.

4.1 Model of floating point computation

The **Standard Computational Model (SCM)** assumes that, for any two floating point numbers χ and ψ, the basic arithmetic operations satisfy the equality

$$[\chi \text{ op } \psi] = (\chi \text{ op } \psi)(1+\epsilon), \qquad |\epsilon| \leq \mathbf{u}, \text{ and } \text{ op } \in \{+,-,*,/\}.$$

The quantity ϵ is a function of χ, ψ and op. Sometimes we add a subscript $(\epsilon_+, \epsilon_*, \ldots)$ to indicate what operation generated the $(1+\epsilon)$ error factor. We always assume that all the input variables to an operation are floating point numbers. **We can interpret the SCM as follows: These operations are performed exactly and it is only in storing the result that a roundoff error occurs (equal to that introduced when a real number is stored as a floating point number).**

Remark 4 *Given* $\chi, \psi \in F$*, performing any operation* $\text{op} \in \{+,-,*,/\}$ *with* χ *and* ψ *in floating point arithmetic,* $[\chi \text{ op } \psi]$*, is a stable operation: Let* $\zeta = \chi \text{ op } \psi$ *and* $\hat{\zeta} = \zeta + \delta\!\zeta = [\chi \;(op)\; \psi]$*. Then* $|\delta\!\zeta| \leq \mathbf{u}|\zeta|$ *and hence* $\hat{\zeta}$ *is close to* ζ *(it has* k *correct binary digits).*

For certain problems it is convenient to use the **Alternative Computational Model (ACM)** [6] which also assumes for the basic arithmetic operations that

$$[\chi \text{ op } \psi] = \frac{\chi \text{ op } \psi}{1+\epsilon}, \qquad |\epsilon| \leq \mathbf{u}, \text{ and } \text{ op } \in \{+,-,*,/\}.$$

As for the standard computation model, the quantity ϵ is a function of χ, ψ and op. Note that the ϵ's produced using the standard and alternative models are generally not equal.

Remark 5 *Notice that the Taylor series expansion of* $1/(1+\epsilon)$ *is given by*

$$\frac{1}{1+\epsilon} = 1 + (-\epsilon) + O(\epsilon^2),$$

which explains how the SCM and ACM are related.

Remark 6 *Sometimes it is more convenient to use the SCM and sometimes the ACM. Trial and error, and eventually experience, will determine which one to use.*

4.2 Stability of a numerical algorithm

In the presence of round-off error, an algorithm involving numerical computations cannot be expected to yield the exact result. Thus, the notion of "correctness" applies only to the execution of algorithms in exact arithmetic. Here we briefly introduce the notion of "stability" of algorithms.

Let $f : \mathcal{D} \to \mathcal{R}$ be a mapping from the domain $\mathcal{D}$ to the range $\mathcal{R}$ and let $\check{f} : \mathcal{D} \to \mathcal{R}$ represent the mapping that captures the execution in floating point arithmetic of a given algorithm which computes f.

The algorithm is said to be **backward stable** if for all $x \in \mathcal{D}$ there exists a perturbed input $\check{x} \in \mathcal{D}$, close to x, such that $\check{f}(x) = f(\check{x})$. In other words, the computed result equals the result obtained when the exact function is applied to slightly perturbed data. The difference between $\check{x}$ and x, $\delta x = \check{x} - x$, is the perturbation to the original input x.

The reasoning behind backward stability is as follows. The input to a function typically has some errors associated with it. Uncertainty may be due to measurement errors when obtaining the input and/or may be the result of converting real numbers to floating point numbers when storing the input on a computer. If it can be shown that an implementation is backward stable, then it has been proved that the result could have been obtained through exact computations performed on slightly corrupted input. Thus, one can think of the error introduced by the implementation as being comparable to the error introduced when obtaining the input data in the first place.

When discussing error analyses, δx, the difference between x and $\check{x}$, is the backward error and the difference $\check{f}(x) - f(x)$ is the forward error. Throughout the remainder of this note we will be concerned with bounding the backward and/or forward errors introduced by the algorithms executed with floating point arithmetic.

The algorithm is said to be **forward stable** on domain $\mathcal{D}$ if for all $x \in \mathcal{D}$ it is that case that $\check{f}(x) \approx f(x)$. In other words, the computed result equals a slight perturbation of the exact result.

4.3 Absolute value of vectors and matrices

In the above discussion of error, the vague notions of "near" and "slightly perturbed" are used. Making these notions exact usually requires the introduction of measures of size for vectors and matrices, i.e., norms. Instead, for the operations analyzed in this note, all bounds are given in terms of the absolute values of the individual elements of the vectors and/or matrices. While it is easy to convert such bounds to bounds involving norms, the converse is not true.

Definition 7 *Given* $x \in \mathbb{R}^n$ *and* $A \in \mathbb{R}^{m \times n}$,

$$|x| = \begin{pmatrix} |\chi_0| \\ |\chi_1| \\ \vdots \\ |\chi_{n-1}| \end{pmatrix} \quad \textit{and} \quad |A| = \begin{pmatrix} |\alpha_{0,0}| & |\alpha_{0,1}| & \ldots & |\alpha_{0,n-1}| \\ |\alpha_{1,0}| & |\alpha_{1,1}| & \ldots & |\alpha_{1,n-1}| \\ \vdots & \vdots & \ddots & \vdots \\ |\alpha_{m-1,0}| & |\alpha_{m-1,1}| & \ldots & |\alpha_{m-1,n-1}| \end{pmatrix}.$$

Definition 8 *Let* $\triangle \in \{<, \leq, =, \geq, >\}$ *and* $x, y \in \mathbb{R}^n$*. Then*

$$|x| \triangle |y| \quad \textit{iff} \quad |\chi_i| \triangle |\psi_i|,$$

with $i = 0, \ldots, n-1$. Similarly, given A and $B \in \mathbb{R}^{m \times n}$,

$$|A| \vartriangle |B| \quad \textit{iff} \quad |\alpha_{ij}| \vartriangle |\beta_{ij}|,$$

with $i = 0, \ldots, m-1$ and $j = 0, \ldots, n-1$.

The next Lemma is exploited in later sections:

Lemma 9 *Let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$. Then $|AB| \leq |A||B|$.*

Exercise 10 *Prove Lemma 9.*

<u>Answer:</u> Let $C = AB$. Then the (i,j) entry in $|C|$ is given by

$$|\gamma_{i,j}| = \left| \sum_{p=0}^{k} \alpha_{i,p} \beta_{p,j} \right| \leq \sum_{p=0}^{k} |\alpha_{i,p} \beta_{p,j}| = \sum_{p=0}^{k} |\alpha_{i,p}||\beta_{p,j}|$$

which equals the (i,j) entry of $|A||B|$. Thus $|AB| \leq |A||B|$.
<u>End of Answer:</u>

The fact that the bounds that we establish can be easily converted into bounds involving norms is a consequence of the following theorem, where $\| \ \|_F$ indicates the Frobenius matrix norm.

Theorem 11 *Let $A, B \in \mathbb{R}^{m \times n}$. If $|A| \leq |B|$ then $\|A\|_1 \leq \|B\|_1$, $\|A\|_\infty \leq \|B\|_\infty$, and $\|A\|_F \leq \|B\|_F$.*

Exercise 12 *Prove Theorem 11.*

4.4 Deriving dense linear algebra algorithms

In various papers, we have shown that for a broad class of linear algebra operations, given an operation one can systematically derive algorithms for computing it [5]. The primary vehicle in the derivation of algorithms is a worksheet to be filled in a prescribed order [2]. We do not discuss the derivation worksheet in this note in order to keep the focus of on the derivation of error analyses. However, we encourage the reader to compare the error worksheet, introduced next, to the worksheet for deriving algorithms, as the order in which the error worksheet is filled mirrors that of the derivation worksheet.

5 Stability of the Dot Product Operation and Introduction to the Error Worksheet

The matrix-vector multiplication algorithm discussed in the next section requires the computation of the dot (inner) product (Dot) of vectors $x, y \in \mathbb{R}^n$: $\kappa := x^T y$. In this section, we give an algorithm for this operation and the related error results. We also introduce the error-worksheet as a framework for presenting the error analysis side-by-side with the algorithm.

5.1 An algorithm for computing Dot

We will consider the algorithm given in Figure 2. It uses the FLAME notation [5, 2] to express the computation

$$\kappa := \Big(\big((\chi_0 \psi_0 + \chi_1 \psi_1) + \cdots \big) + \chi_{n-2} \psi_{n-2} \Big) + \chi_{n-1} \psi_{n-1} \tag{2}$$

in the indicated order.

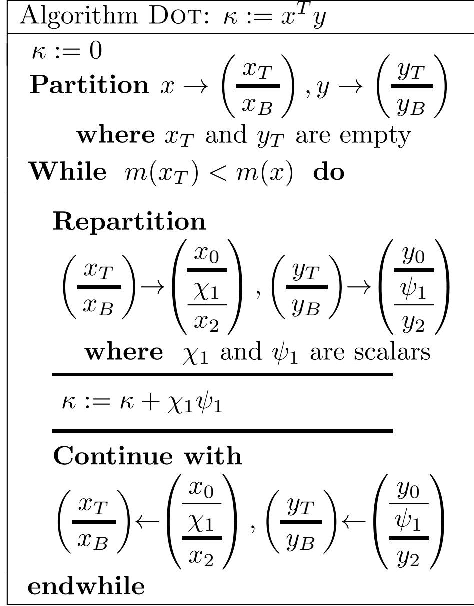

Figure 2: Algorithm for computing $\kappa := x^T y$.

5.2 A simple start

Before giving a general result, let us focus on the case where $n = 2$:

$$\kappa := \chi_0\psi_0 + \chi_1\psi_1.$$

Then, under the computational model given in Section 4, if $\kappa := \chi_0\psi_0 + \chi_1\psi_1$ is executed, the computed result, $\check{\kappa}$, satisfies

$$\begin{aligned}
\check{\kappa} &= \left[\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \right] \\
&= [\chi_0\psi_0 + \chi_1\psi_1] \\
&= [[\chi_0\psi_0] + [\chi_1\psi_1]] \\
&= [\chi_0\psi_0(1+\epsilon_*^{(0)}) + \chi_1\psi_1(1+\epsilon_*^{(1)})] \\
&= (\chi_0\psi_0(1+\epsilon_*^{(0)}) + \chi_1\psi_1(1+\epsilon_*^{(1)}))(1+\epsilon_+^{(1)}) \\
&= \chi_0\psi_0(1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) + \chi_1\psi_1(1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \\
&= \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} \psi_0(1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) \\ \psi_1(1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \end{pmatrix} \\
&= \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}^T \begin{pmatrix} (1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) & 0 \\ 0 & (1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix} \\
&= \begin{pmatrix} \chi_0(1+\epsilon_*^{(0)})(1+\epsilon_+^{(1)}) \\ \chi_1(1+\epsilon_*^{(1)})(1+\epsilon_+^{(1)}) \end{pmatrix}^T \begin{pmatrix} \psi_0 \\ \psi_1 \end{pmatrix},
\end{aligned}$$

where $|\epsilon_*^{(0)}|, |\epsilon_*^{(1)}|, |\epsilon_+^{(1)}| \leq \mathbf{u}$.

Exercise 13 *Repeat the above steps for the computation*

$$\kappa := ((\chi_0\psi_0 + \chi_1\psi_1) + \chi_2\psi_2),$$

computing in the indicated order.

5.3 Preparation

Under the computational model given in Section 4, the computed result of (2), $\check{\kappa}$, satisfies

$$\begin{aligned}\check{\kappa} &= \Bigg(\bigg(\Big(\chi_0\psi_0(1+\epsilon_*^{(0)}) + \chi_1\psi_1(1+\epsilon_*^{(1)})\Big)(1+\epsilon_+^{(1)}) + \cdots\bigg)(1+\epsilon_+^{(n-2)}) \\ &\qquad + \chi_{n-1}\psi_{n-1}(1+\epsilon_*^{(n-1)})\Bigg)(1+\epsilon_+^{(n-1)}) \\ &= \sum_{i=0}^{n-1}\left(\chi_i\psi_i(1+\epsilon_*^{(i)})\prod_{j=i}^{n-1}(1+\epsilon_+^{(j)})\right), \end{aligned} \tag{3}$$

where $\epsilon_+^{(0)} = 0$ and $|\epsilon_*^{(0)}|, |\epsilon_*^{(j)}|, |\epsilon_+^{(j)}| \leq \mathbf{u}$ for $j = 1, \ldots, n-1$.

Clearly, a notation to keep expressions from becoming unreadable is desirable. For this reason we introduce the symbol θ_j:

Lemma 14 *Let* $\epsilon_i \in \mathbb{R}$, $0 \leq i \leq n-1$, $n\mathbf{u} < 1$, *and* $|\epsilon_i| \leq \mathbf{u}$. *Then* $\exists\ \theta_n \in \mathbb{R}$ *such that*

$$\prod_{i=0}^{n-1}(1+\epsilon_i)^{\pm 1} = 1 + \theta_n,$$

with $|\theta_n| \leq n\mathbf{u}/(1-n\mathbf{u})$.

Proof: By Mathematical Induction.

Base case. $n = 1$. Trivial.

Inductive Step. The Inductive Hypothesis (I.H.) tells us that for all $\epsilon_i \in \mathbb{R}$, $0 \leq i \leq n-1$, $n\mathbf{u} < 1$, and $|\epsilon_i| \leq \mathbf{u}$, there exists a $\theta_n \in \mathbb{R}$ such that

$$\prod_{i=0}^{n-1}(1+\epsilon_i)^{\pm 1} = 1 + \theta_n, \text{ with } |\theta_n| \leq n\mathbf{u}/(1-n\mathbf{u}).$$

We will show that if $\epsilon_i \in \mathbb{R}$, $0 \leq i \leq n$, $(n+1)\mathbf{u} < 1$, and $|\epsilon_i| \leq \mathbf{u}$, then there exists a $\theta_{n+1} \in \mathbb{R}$ such that

$$\prod_{i=0}^{n}(1+\epsilon_i)^{\pm 1} = 1 + \theta_{n+1}, \text{ with } |\theta_{n+1}| \leq (n+1)\mathbf{u}/(1-(n+1)\mathbf{u}).$$

Case 1: $\prod_{i=0}^{n}(1+\epsilon_i)^{\pm 1} = \prod_{i=0}^{n-1}(1+\epsilon_i)^{\pm 1}(1+\epsilon_n)$. See Exercise 15.

Case 2: $\prod_{i=0}^{n}(1+\epsilon_i)^{\pm 1} = (\prod_{i=0}^{n-1}(1+\epsilon_i)^{\pm 1})/(1+\epsilon_n)$. By the I.H. there exists a θ_n such that $(1+\theta_n) = \prod_{i=0}^{n-1}(1+\epsilon_i)^{\pm 1}$ and $|\theta_n| \leq n\mathbf{u}/(1-n\mathbf{u})$. Then

$$\frac{\prod_{i=0}^{n-1}(1+\epsilon_i)^{\pm 1}}{1+\epsilon_n} = \frac{1+\theta_n}{1+\epsilon_n} = 1 + \underbrace{\frac{\theta_n - \epsilon_n}{1+\epsilon_n}}_{\theta_{n+1}},$$

which tells us how to pick θ_{n+1}. Now

$$|\theta_{n+1}| = \left| \frac{\theta_n - \epsilon_n}{1 + \epsilon_n} \right| \leq \frac{|\theta_n| + \mathbf{u}}{1 - \mathbf{u}} \leq \frac{\frac{n\mathbf{u}}{1-n\mathbf{u}} + u}{1 - \mathbf{u}} = \frac{n\mathbf{u} + (1 - n\mathbf{u})\mathbf{u}}{(1 - n\mathbf{u})(1 - \mathbf{u})}$$
$$= \frac{(n+1)\mathbf{u} - n\mathbf{u}^2}{1 - (n+1)\mathbf{u} + n\mathbf{u}^2} \leq \frac{(n+1)\mathbf{u}}{1 - (n+1)\mathbf{u}}.$$

By the Principle of Mathematical Induction, the result holds.

Exercise 15 *Complete the proof of Lemma 14.*

The quantity θ_n will be used throughout this note. **It is not intended to be a specific number.** Instead, it is an order of magnitude identified by the subscript n, which indicates the number of error factors of the form $(1 + \epsilon_i)$ and/or $(1 + \epsilon_i)^{-1}$ that are grouped together to form $(1 + \theta_n)$. Since the bound on $|\theta_n|$ occurs often, we assign it a symbol as follows:

Definition 16 *For all $n \geq 1$ and $n\mathbf{u} < 1$, define $\gamma_n := n\mathbf{u}/(1 - n\mathbf{u})$.*

With this notation, (3) simplifies to

$$\begin{aligned}
\check{\kappa} &= \chi_0\psi_0(1 + \theta_n) + \chi_1\psi_1(1 + \theta_n) + \cdots + \chi_{n-1}\psi_{n-1}(1 + \theta_2) && (4)\\
&= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \begin{pmatrix} (1+\theta_n) & 0 & 0 & \cdots & 0 \\ 0 & (1+\theta_n) & 0 & \cdots & 0 \\ 0 & 0 & (1+\theta_{n-1}) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & (1+\theta_2) \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix} && (5)\\
&= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}^T \left(I + \begin{pmatrix} \theta_n & 0 & 0 & \cdots & 0 \\ 0 & \theta_n & 0 & \cdots & 0 \\ 0 & 0 & \theta_{n-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \theta_2 \end{pmatrix} \right) \begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{n-1} \end{pmatrix},
\end{aligned}$$

where $|\theta_j| \leq \gamma_j$, $j = 2, \ldots, n$.

Two instances of the symbol θ_n, appearing even in the same expression, typically do not represent the same number. For example, in (4) a $(1 + \theta_n)$ multiplies each of the terms $\chi_0\psi_0$ and $\chi_1\psi_1$, but these two instances of θ_n, as a rule, do not denote the same quantity. In particular, One should be careful when factoring out such quantities.

As part of the analyses the following bounds will be useful to bound error that accumulates:

Lemma 17 *If $n, b \geq 1$ then $\gamma_n \leq \gamma_{n+b}$ and $\gamma_n + \gamma_b + \gamma_n\gamma_b \leq \gamma_{n+b}$.*

Exercise 18 *Prove Lemma 17.*

5.4 Target result

It is of interest to accumulate the roundoff error encountered during computation as a perturbation of input and/or output parameters:

- $\check{\kappa} = (x + \delta x)^T y$; ($\check{\kappa}$ is the exact output for a slightly perturbed x)
- $\check{\kappa} = x^T(y + \delta y)$; ($\check{\kappa}$ is the exact output for a slightly perturbed y)

- $\check{\kappa} = x^T y + \delta\!\kappa$. (*$\check{\kappa}$ equals the exact result plus an error*)

The first two are backward error results (error is accumulated onto input parameters, showing that the algorithm is numerically stable since it yields the exact output for a slightly perturbed input) while the last one is a forward error result (error is accumulated onto the answer). We will see that in different situations, a different error result may be needed by analyses of operations that require a dot product.

Let us focus on the second result. Ideally one would show that each of the entries of y is slightly perturbed relative to that entry:

$$\delta\!y = \begin{pmatrix} \sigma_0\psi_0 \\ \vdots \\ \sigma_{n-1}\psi_{n-1} \end{pmatrix} = \begin{pmatrix} \sigma_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{n-1} \end{pmatrix} \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = \Sigma y,$$

where each σ_i is "small" and $\Sigma = \mathrm{diag}(\sigma_0, \ldots, \sigma_{n-1})$. The following special structure of Σ, inspired by (5) will be used in the remainder of this note:

$$\Sigma^{(n)} = \begin{cases} 0 \times 0 \text{ matrix} & \text{if } n = 0 \\ \theta_1 & \text{if } n = 1 \\ \mathrm{diag}(\theta_n, \theta_n, \theta_{n-1}, \ldots, \theta_2) & \text{otherwise.} \end{cases} \tag{6}$$

Recall that θ_j is an order of magnitude variable with $|\theta_j| \le \gamma_j$.

Exercise 19 *Let $k \ge 0$ and assume that $|\epsilon_1|, |\epsilon_2| \le \mathbf{u}$, with $\epsilon_1 = 0$ if $k = 0$. Show that*

$$\left(\begin{array}{c|c} I + \Sigma^{(k)} & 0 \\ \hline 0 & (1 + \epsilon_1) \end{array}\right)(1 + \epsilon_2) = (I + \Sigma^{(k+1)}).$$

Hint: reason the case where $k = 0$ separately from the case where $k > 0$.

We state a theorem that captures how error is accumulated by the algorithm.

Theorem 20 *Let $x, y \in \mathbb{R}^n$ and let $\kappa := x^T y$ be computed by executing the algorithm in Figure 2. Then*

$$\check{\kappa} = \left[x^T y\right] = x^T (I + \Sigma^{(n)}) y.$$

5.5 A proof in traditional format

In the below proof, we will pick symbols to denote vectors so that the proof can be easily related to the alternative framework to be presented in Section 5.6.

Proof: By Mathematical Induction on n, the length of vectors x and y.

Base case. $m(x) = m(y) = 0$. Trivial.

Inductive Step. I.H.: Assume that if $x_T, y_T \in \mathbb{R}^k$, $k > 0$, then

$$\left[x_T^T y_T\right] = x_T^T (I + \Sigma_T) y_T, \text{ where } \Sigma_T = \Sigma^{(k)}.$$

We will show that when $x_T, y_T \in \mathbb{R}^{k+1}$, the equality $\left[x_T^T y_T\right] = x_T^T(I+\Sigma_T)y_T$ holds *true* again. Assume that $x_T, y_T \in \mathbb{R}^{k+1}$, and partition $x_T \rightarrow \begin{pmatrix} x_0 \\ \hline \chi_1 \end{pmatrix}$ and $y_T \rightarrow \begin{pmatrix} y_0 \\ \hline \psi_1 \end{pmatrix}$. Then

$$\begin{aligned}
\left[\begin{pmatrix} x_0 \\ \hline \chi_1 \end{pmatrix}^T \begin{pmatrix} y_0 \\ \hline \psi_1 \end{pmatrix}\right] &= \left[\left[x_0^T y_0\right] + \left[\chi_1 \psi_1\right]\right] && \text{(definition)} \\
&= \left[x_0^T(I+\Sigma_0)y_0 + \left[\chi_1\psi_1\right]\right] && \text{(I.H. with } x_T = x_0, \\
& && \quad y_T = y_0, \text{ and } \Sigma_0 = \Sigma^{(k)}) \\
&= \left(x_0^T(I+\Sigma_0)y_0 + \chi_1\psi_1(1+\epsilon_*)\right)(1+\epsilon_+) && \text{(SCM, twice)} \\
&= \begin{pmatrix} x_0 \\ \hline \chi_1 \end{pmatrix}^T \left(\begin{array}{c|c} (I+\Sigma_0) & 0 \\ \hline 0 & (1+\epsilon_*) \end{array}\right)(1+\epsilon_+)\begin{pmatrix} y_0 \\ \hline \psi_1 \end{pmatrix} && \text{(rearrangement)} \\
&= x_T^T(I+\Sigma_T)y_T && \text{(renaming),}
\end{aligned}$$

where $|\epsilon_*|, |\epsilon_+| \leq \mathbf{u}$, $\epsilon_+ = 0$ if $k = 0$, and $(I+\Sigma_T) = \left(\begin{array}{c|c} (I+\Sigma_0) & 0 \\ \hline 0 & (1+\epsilon_*) \end{array}\right)(1+\epsilon_+)$ so that $\Sigma_T = \Sigma^{(k+1)}$.

By the Principle of Mathematical Induction, the result holds.

5.6 A weapon of math induction for the war on error (optional)

We focus the reader's attention on Figure 3 in which we present a framework, which we will call the **error worksheet**, for presenting the inductive proof of Theorem 20 side-by-side with the algorithm for DOT. This framework, in a slightly different form, was first introduced in [1]. The expressions enclosed by { } (in the grey boxes) are predicates describing the state of the variables used in the algorithms and in their analysis. In the worksheet, we use superscripts to indicate the iteration number, thus, the symbols v^i and v^{i+1} do not denote two different variables, but two different states of variable v.

The proof presented in Figure 3 goes hand in hand with the algorithm, as it shows that before and after each iteration of the loop that computes $\kappa := x^T y$, the variables $\check{\kappa}, x_T, y_T, \Sigma_T$ are such that the predicate

$$\{\check{\kappa} = x_T^T(I+\Sigma_T)y_T \wedge k = m(x_T) \wedge \Sigma_T = \Sigma^{(k)}\} \tag{7}$$

holds *true*. This relation is satisfied at each iteration of the loop, so it is also satisfied when the loop completes. Upon completion, the loop guard is $m(x_T) = m(x) = n$, which implies that $\check{\kappa} = x^T(I+\Sigma^{(n)})y$, i.e., the thesis of the theorem, is satisfied too.

In details, the inductive proof of Theorem 20 is captured by the error worksheet as follows:

Base case. In Step 2a, i.e. before the execution of the loop, predicate (7) is satisfied, as $k = m(x_T) = 0$.

Inductive step. Assume that the predicate (7) holds *true* at Step 2b, i.e., at the top of the loop. Then Steps 6, 7, and 8 in Figure 3 prove that the predicate is satisfied again at Step 2c, i.e., the bottom of the loop. Specifically,

- Step 6 holds by virtue of the equalities $x_0 = x_T, y_0 = y_T$, and $\Sigma_0^i = \Sigma_T$.
- The update in Step 8-left introduces the error indicated in Step 8-right (SCM, twice), yielding the results for Σ_0^{i+1} and σ_1^{i+1}, leaving the variables in the state indicated in Step 7.
- Finally, the redefinition of Σ_T in Step 5b transforms the predicate in Step 7 into that of Step 2c, completing the inductive step.

By the Principle of Mathematical Induction, the predicate (7) holds for all iterations. In particular, when the loop terminates, the predicate becomes

$$\check{\kappa} = x^T(I+\Sigma^{(n)})y \wedge n = m(x_T).$$

	Error side	Step
$\kappa := 0$	$\{ \Sigma = 0 \}$	1a
Partition $x \rightarrow \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right), y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right)$, **where** x_T and y_T are empty, and Σ_T is 0×0	$\Sigma \rightarrow \left(\begin{array}{c\|c} \Sigma_T & 0 \\ \hline 0 & \Sigma_B \end{array}\right)$	4
	$\{\check{\kappa} = x_T^T(I + \Sigma_T)y_T \wedge \Sigma_T = \Sigma^{(k)} \wedge m(x_T) = k\}$	2a
While $m(x_T) < m(x)$ **do**		3
	$\{\check{\kappa} = x_T^T(I + \Sigma_T)y_T \wedge \Sigma_T = \Sigma^{(k)} \wedge m(x_T) = k\}$	2b
Repartition $\left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \rightarrow \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ x_2 \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_0 \\ \hline \psi_1 \\ y_2 \end{array}\right)$, **where** χ_1, ψ_1, and σ_1^i are scalars	$\left(\begin{array}{c\|c} \Sigma_T & 0 \\ \hline 0 & \Sigma_B \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} \Sigma_0^i & 0 & 0 \\ \hline 0 & \sigma_1^i & 0 \\ 0 & 0 & \Sigma_2 \end{array}\right)$	5a
	$\{\check{\kappa}^i = x_0^T(I + \Sigma_0^i)y_0 \wedge \Sigma_0^i = \Sigma^{(k)} \wedge m(x_0) = k\}$	6
$\kappa := \kappa + \chi_1\psi_1$	$\check{\kappa}^{i+1} = \left(\check{\kappa}^i + \chi_1\psi_1(1 + \epsilon_*)\right)(1 + \epsilon_+)$ SCM, twice ($\epsilon_+ = 0$ if $k = 0$) $= \left(x_0^T(I + \Sigma_0^{(k)})y_0 + \chi_1\psi_1(1 + \epsilon_*)\right)(1 + \epsilon_+)$ Step 6: I.H. $= \left(\begin{array}{c} x_0 \\ \chi_1 \end{array}\right)^T \left(\begin{array}{c\|c} I + \Sigma_0^{(k)} & 0 \\ \hline 0 & 1 + \epsilon_* \end{array}\right) (1 + \epsilon_+) \left(\begin{array}{c} y_0 \\ \psi_1 \end{array}\right)$ Rearrange $= \left(\begin{array}{c} x_0 \\ \chi_1 \end{array}\right)^T \left(I + \Sigma^{(k+1)}\right) \left(\begin{array}{c} y_0 \\ \psi_1 \end{array}\right)$ Exercise 19	8
	$\left\{\begin{array}{l} \check{\kappa}^{i+1} = \left(\begin{array}{c} x_0 \\ \chi_1 \end{array}\right)^T \left(I + \left(\begin{array}{c\|c} \Sigma_0^{i+1} & 0 \\ \hline 0 & \sigma_1^{i+1} \end{array}\right)\right) \left(\begin{array}{c} y_0 \\ \psi_1 \end{array}\right) \\ \wedge \left(\begin{array}{c\|c} \Sigma_0^{i+1} & 0 \\ \hline 0 & \sigma_1^{i+1} \end{array}\right) = \Sigma^{(k+1)} \wedge m\left(\begin{array}{c} x_0 \\ \chi_1 \end{array}\right) = (k+1) \end{array}\right\}$	7
Continue with $\left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \leftarrow \left(\begin{array}{c} x_0 \\ \chi_1 \\ \hline x_2 \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ \hline y_2 \end{array}\right)$,	$\left(\begin{array}{c\|c} \Sigma_T & 0 \\ \hline 0 & \Sigma_B \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} \Sigma_0^{i+1} & 0 & 0 \\ 0 & \sigma_1^{i+1} & 0 \\ \hline 0 & 0 & \Sigma_2 \end{array}\right)$	5b
	$\{\check{\kappa} = x_T^T(I + \Sigma_T)y_T \wedge \Sigma_T = \Sigma^{(k)} \wedge m(x_T) = k\}$	2c
endwhile		
	$\{\check{\kappa} = x_T^T(I + \Sigma_T)y_T \wedge \Sigma_T = \Sigma^{(k)} \wedge m(x_T) = k \wedge m(x_T) = m(x)\}$	2d
	$\{\check{\kappa} = x^T(I + \Sigma^{(n)})y \wedge m(x) = n\}$	1b

Figure 3: Error worksheet completed to establish the backward error result for the given algorithm that computes the Dot operation.

This completes the discussion of the proof as captured by Figure 3.

In the derivation of algorithms, the concept of **loop-invariant** plays a central role. Let $\mathcal{L}$ be a loop and $\mathcal{P}$ a predicate. If $\mathcal{P}$ is *true* before the execution of $\mathcal{L}$, at the beginning and at the end of each iteration of $\mathcal{L}$, and after the completion of $\mathcal{L}$, then predicate $\mathcal{P}$ is a *loop-invariant* with respect to $\mathcal{L}$. Similarly, we give the definition of **error-invariant**.

Definition 21 *We call the predicate involving the operands and error operands in Steps 2a–d the* **error-invariant** *for the analysis. This predicate is* true *before and after each iteration of the loop.*

For any algorithm, the loop-invariant and the error-invariant are related in that the former describes the status of the computation at the beginning and the end of each iteration, while the latter captures an error result for the computation indicated by the loop-invariant.

The reader will likely think that the error worksheet is an overkill when proving the error result for the dot product. We agree. However, it links a proof by induction to the execution of a loop, which we believe is useful. Elsewhere, as more complex operations are analyzed, the benefits of the structure that the error

worksheet provides will become more obvious. (We will analyze more complex algorithms as the course proceeds.)

5.7 Results

A number of useful consequences of Theorem 20 follow. These will be used later as an inventory (library) of error results from which to draw when analyzing operations and algorithms that utilize Dot.

Corollary 22 *Under the assumptions of Theorem 20 the following relations hold:*

R1-B: (Backward analysis) $\check{\kappa} = (x + \delta x)^T y$, *where* $|\delta x| \leq \gamma_n |x|$, *and* $\check{\kappa} = x^T(y + \delta y)$, *where* $|\delta y| \leq \gamma_n |y|$;

R1-F: (Forward analysis) $\check{\kappa} = x^T y + \delta\kappa$, *where* $|\delta\kappa| \leq \gamma_n |x|^T |y|$.

Proof: We leave the proof of R1-B as an exercise. For R1-F, let $\delta\kappa = x^T \Sigma^{(n)} y$, where $\Sigma^{(n)}$ is as in Theorem 20. Then

$$\begin{aligned} |\delta\kappa| &= |x^T \Sigma^{(n)} y| \\ &\leq |\chi_0||\theta_n||\psi_0| + |\chi_1||\theta_n||\psi_1| + \cdots + |\chi_{n-1}||\theta_2||\psi_{n-1}| \\ &\leq \gamma_n|\chi_0||\psi_0| + \gamma_n|\chi_1||\psi_1| + \cdots + \gamma_2|\chi_{n-1}||\psi_{n-1}| \\ &\leq \gamma_n |x|^T |y|. \end{aligned}$$

Exercise 23 *Prove R1-B.*

6 Stability of a Matrix-Vector Multiplication Algorithm

In this section, we discuss the numerical stability of the specific matrix-vector multiplication algorithm that computes $y := Ax$ via dot products. This allows us to show how results for the dot product can be used in the setting of a more complicated algorithm.

6.1 An algorithm for computing Gemv

We will consider the algorithm given in Figure 4 for computing $y := Ax$, which computes y via dot products.

6.2 Analysis

Assume $A \in \mathbb{R}^{m \times n}$ and partition

$$A = \begin{pmatrix} a_0^T \\ a_1^T \\ \vdots \\ a_{m-1}^T \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix}.$$

Then

$$\begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} := \begin{pmatrix} a_0^T x \\ a_1^T x \\ \vdots \\ a_{m-1}^T x \end{pmatrix}.$$

Algorithm Gemv: $y := Ax$

$y := 0$
Partition $A \rightarrow \left(\frac{A_T}{A_B}\right)$, $y \rightarrow \left(\frac{y_T}{y_B}\right)$
where A_T and y_T are empty
While $m(A_T) < m(A)$ **do**
Repartition
$\left(\frac{A_T}{A_B}\right) \rightarrow \begin{pmatrix} A_0 \\ \hline a_1^T \\ \hline A_2 \end{pmatrix}$, $\left(\frac{y_T}{y_B}\right) \rightarrow \begin{pmatrix} y_0 \\ \hline \psi_1 \\ \hline y_2 \end{pmatrix}$
where a_1^T is a row and ψ_1 is a scalar
$\psi_1 := a_1^T x$
Continue with
$\left(\frac{x_T}{x_B}\right) \leftarrow \begin{pmatrix} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{pmatrix}$, $\left(\frac{y_T}{y_B}\right) \leftarrow \begin{pmatrix} y_0 \\ \hline \psi_1 \\ \hline y_2 \end{pmatrix}$
endwhile

Figure 4: Algorithm for computing $y := Ax$.

From Corollary 22 R1-B regarding the dot product we know that

$$\check{y} = \begin{pmatrix} \check{\psi}_0 \\ \check{\psi}_1 \\ \vdots \\ \check{\psi}_{m-1} \end{pmatrix} = \begin{pmatrix} (a_0 + \delta a_0)^T x \\ (a_1 + \delta a_1)^T x \\ \vdots \\ (a_{m-1} + \delta a_{m-1})^T x \end{pmatrix} = \left(\begin{pmatrix} a_0^T \\ a_1^T \\ \vdots \\ a_{m-1}^T \end{pmatrix} + \begin{pmatrix} \delta a_0^T \\ \delta a_1^T \\ \vdots \\ \delta a_{m-1}^T \end{pmatrix} \right) x = (A + \Delta A)x,$$

where $|\delta a_i| \leq \gamma_n |a_i|$, $i = 0, \ldots, m-1$, and hence $|\Delta A| \leq \gamma_n |A|$. .

Also, from Corollary 22 R1-F regarding the dot product we know that

$$\check{y} = \begin{pmatrix} \check{\psi}_0 \\ \check{\psi}_1 \\ \vdots \\ \check{\psi}_{m-1} \end{pmatrix} = \begin{pmatrix} a_0^T x + \delta\psi_0 \\ a_1^T x + \delta\psi_1 \\ \vdots \\ a_{m-1}^T x + \delta\psi_{m-1} \end{pmatrix} = \begin{pmatrix} a_0^T \\ a_1^T \\ \vdots \\ a_{m-1}^T \end{pmatrix} x + \begin{pmatrix} \delta\psi_0 \\ \delta\psi_1 \\ \vdots \\ \delta\psi_{m-1} \end{pmatrix} = Ax + \delta y.$$

where $|\delta\psi_i| \leq \gamma_n |a_i|^T |x|$ and hence $|\delta y| \leq \gamma_n |A||x|$.

The above observations can be summarized in the following theorem:

Theorem 24 Error results for matrix-vector multiplication. *Let $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ and consider the assignment $y := Ax$ implemented via the algorithm in Figure 4. Then these equalities hold:*

R1-B: $\check{y} = (A + \Delta A)x$, where $|\Delta A| \leq \gamma_n |A|$.

R2-F: $\check{y} = Ax + \delta y$, where $|\delta y| \leq \gamma_n |A||x|$.

Exercise 25 *In the above theorem, could one instead prove the result*

$$\check{y} = A(x + \delta x),$$

where δx is "small"?

Algorithm Gemm: $C := AB$

$C := 0$
Partition $C \rightarrow \left(C_L \,\|\, C_R \right), B \rightarrow \left(B_L \,\|\, B_R \right)$
where C_L and B_L are empty
While $n(C_L) < n(C)$ **do**
Repartition
$\left(C_L \,\|\, C_R \right) \rightarrow \left(C_0 \,\|\, c_1 \,\|\, C_2 \right), \left(B_L \,\|\, B_R \right) \rightarrow \left(B_0 \,\|\, b_1 \,\|\, B_2 \right)$
where c_1 and b_1 are columns
$c_1 := Ab_1$
Continue with
$\left(C_L \,\|\, C_R \right) \leftarrow \left(C_0 \,\|\, c_1 \,\|\, C_2 \right), \left(B_L \,\|\, B_R \right) \leftarrow \left(B_0 \,\|\, b_1 \,\|\, B_2 \right)$
endwhile

Figure 5: Algorithm for computing $C := AB$ one column at a time.

7 Stability of a Matrix-Matrix Multiplication Algorithm

In this section, we discuss the numerical stability of the specific matrix-matrix multiplication algorithm that computes $C := AB$ via the matrix-vector multiplication algorithm from the last section, where $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$.

7.1 An algorithm for computing Gemm

We will consider the algorithm given in Figure 5 for computing $C := AC$, which computes one column at a time so that the matrix-vector multiplication algorithm from the last section can be used.

7.2 Analysis

Partition

$$C = \left(c_0 \,|\, c_1 \,|\, \cdots \,|\, c_{n-1} \right) \quad \text{and} \quad B = \left(b_0 \,|\, b_1 \,|\, \cdots \,|\, b_{n-1} \right).$$

Then

$$\left(c_0 \,|\, c_1 \,|\, \cdots \,|\, c_{n-1} \right) := \left(Ab_0 \,|\, Ab_1 \,|\, \cdots \,|\, Ab_{n-1} \right).$$

From Corollary 22 R1-B regarding the dot product we know that

$$\begin{aligned} \left(\check{c}_0 \,|\, \check{c}_1 \,|\, \cdots \,|\, \check{c}_{n-1} \right) &= \left(Ab_0 + \delta c_0 \,|\, Ab_1 + \delta c_1 \,|\, \cdots \,|\, Ab_{n-1} + \delta c_{n-1} \right) \\ &= \left(Ab_0 \,|\, Ab_1 \,|\, \cdots \,|\, Ab_{n-1} \right) + \left(\delta c_0 \,|\, \delta c_1 \,|\, \cdots \,|\, \delta c_{n-1} \right) \\ &= AB + \Delta C. \end{aligned}$$

where $|\delta c_j| \leq \gamma_k |A||b_j|$, $j = 0, \ldots, n-1$, and hence $|\Delta C| \leq \gamma_k |A||B|$.

The above observations can be summarized in the following theorem:

Theorem 26 (Forward) error results for matrix-matrix multiplication. *Let $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$ and consider the assignment $C := AB$ implemented via the algorithm in Figure 5. Then the following equality holds:*

R1-F: $\check{C} = AB + \Delta C$, where $|\Delta C| \leq \gamma_k |A||B|$.

Exercise 27 *In the above theorem, could one instead prove the result*

$$\check{C} = (A + \Delta A)(B + \Delta B),$$

where ΔA and ΔB are "small"?

7.3 An application

A collaborator of ours recently implemented a matrix-matrix multiplication algorithm and wanted to check if it gave the correct answer. To do so, he followed the following steps:

- He created random matrices $A \in \mathbb{R}^{m\times k}$, and $C \in \mathbb{R}^{m\times n}$, with positive entries in the range $(0,1)$.
- He computed $C = AB$ with an implementation that was known to be "correct" and assumed it yields the exact solution. (Of course, it has error in it as well. We discuss how he compensated for that, below.)
- He computed $\check{C} = AB$ with his new implementation.
- He computed $\Delta C = \check{C} - C$ and checked that each of its entries satisfied $\delta\gamma_{i,j} \leq 2k\mathbf{u}\gamma_{i,j}$.
- In the above, he took advantage of the fact that A and B had positive entries so that $|A||B| = AB = C$. He also approximated $\gamma_k = \frac{k\mathbf{u}}{1-k\mathbf{u}}$ with $k\mathbf{u}$, and introduced the factor 2 to compensate for the fact that C itself was inexactly computed.

References

[1] Paolo Bientinesi. *Mechanical Derivation and Systematic Analysis of Correct Linear Algebra Algorithms.* PhD thesis, Department of Computer Sciences, The University of Texas, 2006. Technical Report TR-06-46. September 2006.

[2] Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Transactions on Mathematical Software*, 31(1):1–26, March 2005.

[3] Paolo Bientinesi and Robert A. van de Geijn. The science of deriving stability analyses. FLAME Working Note #33. Technical Report AICES-2008-2, Aachen Institute for Computational Engineering Sciences, RWTH Aachen, November 2008.

[4] Paolo Bientinesi and Robert A. van de Geijn. Goal-oriented and modular stability analysis. 32(1):286–308, 2011.

[5] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment. *ACM Trans. Math. Soft.*, 27(4):422–455, December 2001.

[6] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.