Numerical Stability in Gaussian Elimination

John Urschel

In some sense, numerical analysis is a very old field: it involves the study of algorithms for problems in continuous mathematics, and algorithms have been around at least since the Babylonians wrote them on tablets more than four thousand years ago. But with the advent of the computer age in the 1940s, which made numerical calculations on a scale that was previously unimaginable not only possible but easy, understanding the stability of these algorithms became increasingly urgent. What kind of errors could creep in? What were the relative costs of accuracy and speed?

John von Neumann and Herman Goldstine, two of the architects of the modern computer, saw the need for numerical analysis instantly. At the Institute for Advanced Study, they developed the IAS Machine (Figure 1), one of the first[1] stored-program digital computers, and sought to understand the effect of rounding in computations on it. In their seminal 1947 paper "Numerical Inverting of Matrices of High Order," von Neumann and Goldstine studied the stability of one of the oldest and most fundamental known algorithms, Gaussian elimination [vNG47]. Goldstine later wrote [Gol93]:

> Indeed, von Neumann and I chose this topic for the first modern paper on numerical analysis ever written precisely because we viewed the topic as being absolutely basic to numerical mathematics.

Gaussian elimination is very simple—so simple that the "elimination method" is taught to grade school students to solve systems of two or three equations. It is a straightforward way of solving n linear algebraic equations in n unknowns. The earliest variant appears in the eighth chapter of the ancient Chinese text *The Nine Chapters on the Mathematical Art* (Figure 2). And yet, despite being more than two thousand years old and the most popular method for solving a linear system, some aspects of Gaussian elimination are still not understood. Goldstine and von Neumann took on the question of round-off error analysis for matrix factorization and inversion methods in part because of disagreements about the precision of the technique in certain cases. (Alan Turing also presented an analysis around the same time.)

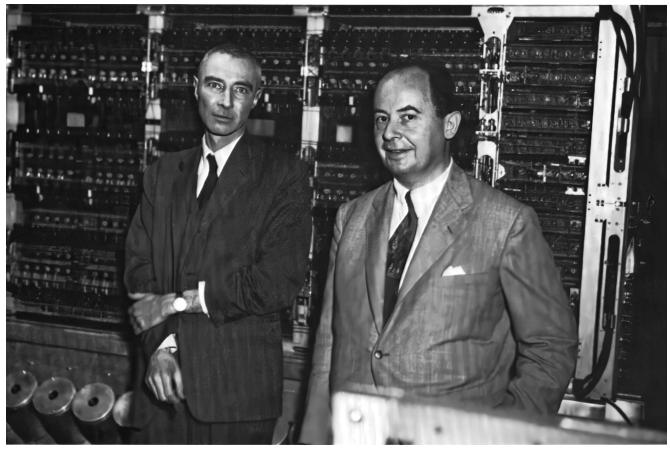

Figure 1. John von Neumann and the director of the Institute for Advanced Study, Robert Oppenheimer, in front of the IAS Machine.

Many mathematicians, including von Neumann himself, were quite pessimistic about the stability of the algorithm.[2] Von Neumann famously changed his opinion only a year later and, together with Goldstine, provided the first framework for understanding it. However, their concrete results were limited, proving stability only in the special case of a symmetric positive definite matrix using complete pivoting.[3] Despite the urgency of pinpointing and avoiding cascading rounding errors as computers

John Urschel is an assistant professor of mathematics at MIT. His email address is urschel@mit.edu.

Communicated by Notices *Associate Editor Emilie Purvine.*

DOI: https://doi.org/10.1090/noti3191

[1] *See also the Small-Scale Experimental Machine (SSEM), completed in 1948 at the University of Manchester, the Electronic Delay Storage Automatic Calculator (EDSAC), completed in 1949 at the University of Cambridge, and the Electronic Discrete Variable Automatic Computer (EDVAC), completed in 1949 at the University of Pennsylvania and fully operational by 1952. All of these machines were inspired by the architecture described by von Neumann in the controversial "First Draft of a Report on the EDVAC," commonly referred to as the von Neumann architecture.*

[2] *See Hotelling's 1943 paper* [Hot43] *and Bargmann, Montgomery, and von Neumann's 1946 technical report* [BMvN46].

[3] *"We have not so far been able to obtain satisfactory error estimates for the pseudo-operational equivalent of the elimination method in its general form... We did, however, succeed in securing everything that is needed in the special case of a definite A"* [vNG47].

$$\left[\begin{array}{ccc|c} 3 & 2 & 1 & 39 \\ 2 & 3 & 1 & 34 \\ 1 & 2 & 3 & 26 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 3 & 2 & 1 & 39 \\ 6 & 9 & 3 & 102 \\ 3 & 6 & 9 & 78 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 3 & 2 & 1 & 39 \\ 0 & 5 & 1 & 24 \\ 0 & 4 & 8 & 39 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 3 & 2 & 1 & 39 \\ 0 & 5 & 1 & 24 \\ 0 & 20 & 40 & 195 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 3 & 2 & 1 & 39 \\ 0 & 5 & 1 & 24 \\ 0 & 0 & 36 & 99 \end{array}\right]$$

Figure 2. The earliest variant of Gaussian elimination appears in the eighth chapter of the ancient Chinese text *The Nine Chapters on the Mathematical Art*. The above procedure is presented to solve the following [Har11]:

> Given 3 bundles of superior paddy [unhusked rice], 2 bundles of ordinary paddy, and 1 bundle of inferior paddy, [together they yield] 39 dou of grain; 2 bundles of superior paddy, 3 bundles of ordinary paddy, and 1 bundle of inferior paddy [together yield] 34 dou of grain; 1 bundle of superior paddy, 2 bundles of ordinary paddy, and 3 bundles of inferior paddy [together yield] 26 dou of grain. Problem: 1 bundle of superior, ordinary, and inferior paddy each yield how much grain?

became widely available, questions regarding the stability of Gaussian elimination remained. In many ways, it's Wilkinson who is the hero of this story, as it was not until his 1961 paper "Error Analysis of Direct Methods of Matrix Inversion" that a more complete analysis of the backward error in Gaussian elimination due to rounding errors was conducted [Wil61]. Most of the open questions that remain involve mathematically quantifying what Wilkinson, even in the 1960s, felt ought to be true.

However, before we delve much further into the topic, allow me to remind the reader what exactly Gaussian elimination is.

What is Gaussian elimination? Given an invertible matrix A and compatible vector b, Gaussian elimination is an algorithm to solve the linear system $Ax = b$ for x in two stages:

(1) factor A into the product $A = LU$ of a lower unitriangular matrix L and an upper triangular matrix U (LU factorization)

$$\begin{matrix} A \\ \end{matrix} \qquad \begin{matrix} L \\ \end{matrix} \qquad \begin{matrix} U \\ \end{matrix}$$
$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \times & 1 & 0 \\ \times & \times & 1 \end{bmatrix} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}$$

(2) solve the linear systems $Ly = b$ for y (forward substitution) and $Ux = y$ for x (backward substitution).

The second stage is straightforward and numerically stable conditional on the stability of the first, and so we can devote our attention to the LU factorization.

The algorithmic procedure for computing this factorization can be concisely summarized by the block formula

$$A = \begin{bmatrix} \alpha & y^T \\ x & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{x}{\alpha} & I \end{bmatrix} \begin{bmatrix} \alpha & y^T \\ 0 & B - \frac{xy^T}{\alpha} \end{bmatrix}, \tag{1}$$

where α is a scalar (called the pivot), x, y are vectors, and B is a matrix. If $B - \frac{xy^T}{\alpha} = L_1 U_1$ for a lower unitriangular L_1 and upper triangular U_1, then the LU factorization of A is given by

$$A = \begin{bmatrix} \alpha & y^T \\ x & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{x}{\alpha} & L_1 \end{bmatrix} \begin{bmatrix} \alpha & y^T \\ 0 & U_1 \end{bmatrix}.$$

This iterative procedure of rank-one updates is essentially what a computer does when performing Gaussian elimination, up to one crucial detail: What to do when the pivot $\alpha = 0$? In this case, some permutations of the rows and/or columns of A are needed to put a nonzero entry in the pivot's place. In reality, a pivot equalling zero is not the only difficulty computers contend with. Very small pivots can be just as dangerous, and the prospect of errors compounding rapidly can also lead to wildly incorrect results. Some pivoting strategy is required to avoid these situations (a topic we will leave for later).

It's worth noting that Gaussian elimination has a number of equivalent mathematical representations, each of which can prove useful in analysis. For example, the block submatrices iteratively created during this algorithm can be tied to the original matrix through a Schur complement, i.e., Gaussian elimination can be equivalently represented by a sequence of $k \times k$ matrices $A^{(k)}$ for k equals n to 1 satisfying $A^{(n)} = A$ and

$$A^{(k)} = Z_k - Y_k W_k^{-1} X_k, \tag{2}$$

where

$$A = \begin{matrix} \\ n-k \\ k \end{matrix} \begin{matrix} \begin{matrix} n-k & k \end{matrix} \\ \begin{bmatrix} W_k & X_k \\ Y_k & Z_k \end{bmatrix} \end{matrix}.$$

In this representation, the resulting LU factorization of A is encoded in the first row and column of each of the iterates $A^{(k)}$. This procedure can also be represented entirely through the language of determinants. Because the determinant of a triangular matrix is the product of its diagonal entries, each entry of L and U is equal to a ratio of minors (determinants of submatrices) of A. The "right" representation of the algorithm largely depends on the specific property/setting being studied.

Gaussian elimination is quite an amazing algorithm. It's not only the oldest in linear algebra, but it is also still the most popular method for numerically solving an unstructured linear system. And it has the benefit of being one of the few linear algebra algorithms that works over an arbitrary field. (The QR factorization, for instance, requires the notion of an inner product.)

Here, in this article, I'd like to focus on this algorithm through the lens of computation, in particular its

numerical stability, and journey through both the mathematical discoveries that have occurred and the questions that remain since the early work of von Neumann, Wilkinson, and others over 60 years ago. However, to do so, we first need to take a detour into the fundamentals of numerical computation.

Floating point arithmetic. To the untrained eye, at this point a natural question may arise:

> Why discuss error at all when the Gaussian elimination algorithm is exact?

While the LU factorization of a rational matrix is indeed rational, the lengths of the matrix elements may be prohibitively long for fast computation in practice.

As an example, consider solving a 1000×1000 linear system with random binary coefficients in the Julia programming language:

```
julia> n = 1000;
       A = rand(Bool,n,n);
       b = rand(Bool,n);
```

First, we solve $Ax = b$ by using 64 bits per number:

```
julia> @time x̂ = Float64.(A) \
       Float64.(b);
  0.01 seconds
```

Next, we solve $Ax = b$ exactly, without any rounding:

```
julia> @time x = RationalBigInt.(A) \
       RationalBigInt.(b);
  2389.89 seconds
```

and output the error from our first computation:

```
julia> norm(x - x̂) / norm(x)
  4.34e-13
```

The first computation took only 10 milliseconds and gave an answer accurate to $\approx 10^{-13}$, while the second took nearly 40 minutes! This is due to the growing complexity of the numbers encountered. For instance, the numerator and denominator of the first entry of x are both over 3200 bits long!

For this reason, in practice, numbers are only approximately stored, using what is called a floating point number system $\mathbb{F}$ (essentially an adaptation of scientific notation tailored for computers with precision constraints).

This system is parameterized by four numbers $\beta, t \in \mathbb{N}$, $e_{\min}, e_{\max} \in \mathbb{Z}$, and consists of all numbers of the form

$$\mathbb{F} = \left\{ \pm m \times \beta^{e-t} \,\middle|\, \begin{matrix} m, e \in \mathbb{Z}, 0 \le m \le \beta^t - 1, \\ e \in [e_{\min}, e_{\max}] \end{matrix} \right\}.$$

See Figure 3 for an example of the bit string representation of 0.15625 in IEEE single precision and Figure 4 for a visual representation of the elements of $\mathbb{F}$ for a toy model with $\beta = 2$, $t = 3$, $e_{\min} = -1$, and $e_{\max} = 3$.

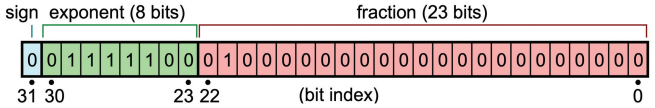

Figure 3. The bit string for $0.15625 = \frac{1}{8} + \frac{1}{32}$ in single precision (32 bits). The number is represented as $(1 + \frac{1}{4}) \times 2^{-3}$ (note $1 + \frac{1}{4} = 1.01_2$). The exponent bias is 127, so the stored exponent is $01111100_2 = 124$.

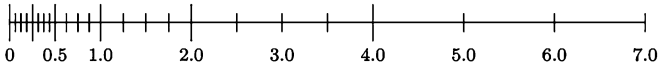

Figure 4. The nonnegative floating point numbers in a toy model with $\beta = 2$, $t = 3$, and $e \in [-1, 3]$. Image appears in Nick Higham's *What Is* series.

When a number $x \in \mathbb{R}$ is input, it is stored as $\mathrm{fl}(x) \in \mathbb{F}$, where $\mathrm{fl} : \mathbb{R} \to \mathbb{F}$ rounds real numbers to the nearest element in $\mathbb{F}$. The effects of rounding when inputting numbers or performing computations can be effectively bounded based on the unit round-off $\mathbf{u} = \beta^{1-t}$, the distance between 1 and the next element of $\mathbb{F}$. In particular, ignoring issues of overflow and underflow, for any $x \in \mathbb{R}$,

$$\mathrm{fl}(x) = x(1+\delta) \qquad \text{for some } |\delta| \le \mathbf{u}.$$

An analogous property also holds for fundamental computer operations: given $x, y \in \mathbb{F}$ and a standard operation op (e.g., $+, -, \times, \div$),

$$\mathrm{fl}(x \operatorname{op} y) = (x \operatorname{op} y)(1+\delta) \qquad \text{for some } |\delta| \le \mathbf{u}.$$

This property is sometimes referred to as the fundamental axiom of floating point arithmetic.

Numerical stability and an elegant mathematical problem. Suppose someone wanted to approximate a function f at a point x using an algorithm $\hat{f}$. How should they judge how well they did? One measure would simply be the distance between the exact and computed solutions $|f(x) - \hat{f}(x)|$ (called forward error). However, if the derivative of f was very large, then this might not be the most fair measure of a good algorithm, as a small change to the input data x could drastically change the output $f(x)$. Instead of asking whether the computed solution is almost the right answer, we could consider whether it is the right answer to almost the right question, i.e., does there exist an $\hat{x}$ such that $\hat{f}(x) = f(\hat{x})$ and $|x - \hat{x}|$ is small? This is called backward error, and though this measure is also imperfect,[4] in this language we can quickly measure the stability of Gaussian elimination.

Thinking of an LU factorization as a function from A to the pair L and U, i.e., $f(A) = (L, U)$, this procedure is backward stable if the computed LU factorization $\hat{f}(A) = (\hat{L}, \hat{U})$

[4] *Consider the outer product of two vectors x and y, i.e., $f(x, y) = xy^T$. The approximation $\hat{f}(x, y) = \mathrm{fl}(xy^T)$ is a very good one, but it is not backward stable! This is because $\mathrm{fl}(xy^T)$ will likely have rank greater than one.*

satisfies $\hat{L}\hat{U} = A + H$ for some H that's not too big. Let $|A|$ denote the matrix with the absolute values of the entries, i.e., $|A|_{ij} = |A_{ij}|$, and consider the matrix inequality $A \leq B$ componentwise. Some version of the following theorem[5] appears in many numerical analysis texts:

Theorem 1 ([GVL13]). *Let $A \in \mathbb{F}^{n\times n}$. If no zero pivots are encountered during Gaussian elimination, then the computed lower and upper triangular matrices $\hat{L}, \hat{U} \in \mathbb{F}^{n\times n}$ satisfy $\hat{L}\hat{U} = A + H$, where*

$$|H| \leq 2(n-1)\mathbf{u}(|A| + |\hat{L}||\hat{U}|) + O(\mathbf{u}^2).$$

While I won't reproduce the proof, I'd like to give some intuition about how these types of results can be proven. One idea is to proceed by induction and consider a single step of Gaussian elimination from

$$A = \begin{bmatrix} \alpha & y^T \\ x & B \end{bmatrix} \quad \text{to} \quad \begin{bmatrix} \alpha & y^T \\ 0 & B - \frac{xy^T}{\alpha} \end{bmatrix}.$$

In floating point arithmetic, this procedure $\mathrm{fl}(B - \mathrm{fl}(\mathrm{fl}(x/\alpha) \times y^T))$ involves three steps, and therefore introduces three errors of relative size at most $\mathbf{u}$. Assuming that the theorem statement holds for matrices of dimension $n-1$, i.e., the computed $\hat{L}_1$ and $\hat{U}_1$ for $\mathrm{fl}(B - \mathrm{fl}(\mathrm{fl}(x/\alpha) \times y^T))$ also satisfies the statement, the result follows pretty quickly, as

$$\hat{L} = \begin{bmatrix} 1 & 0 \\ \mathrm{fl}(x/\alpha) & \hat{L}_1 \end{bmatrix} \quad \text{and} \quad \hat{U} = \begin{bmatrix} \alpha & y^T \\ 0 & \hat{U}_1 \end{bmatrix}.$$

The big takeaway from Theorem 1 is that the stability of Gaussian elimination in floating point arithmetic can be estimated by a simple mathematical quantity: the largest entry in $\hat{L}$ or $\hat{U}$ (scaled by A). It is such a central quantity that we give it a name, the *growth factor*:[6]

$$g(A) := \max\left\{ \|L\|_{\max}, \frac{\|U\|_{\max}}{\|A\|_{\max}} \right\},$$

where $\|\cdot\|_{\max}$ denotes the maximum magnitude entry of a matrix. Theorem 1 converts the practical question of the stability of this algorithm in computation to a purely mathematical one regarding the size of the entries of L and U relative to A. The game, so to speak, is to choose a strategy for permuting rows and columns of A, say with permutation matrices P, Q, so that (1) the strategy can be implemented quickly, and (2) the resulting factorization $PAQ = LU$ has a small growth factor $g(PAQ)$.

[5] *A similar result also holds for the final solution $\hat{x}$ computed from A and b using Gaussian elimination.*

[6] *The exact definition of the growth factor varies from reference to reference. The quantities $\max_k \|A^{(k)}\|_{\max}/\|A\|_{\max}$ and the pair $\|L\|_\infty$ and $\|U\|_\infty/\|A\|_\infty$ are two popular alternatives. They are all equivalent in the sense that they can all be used to bound the backward error of the LU factorization in floating point arithmetic.*

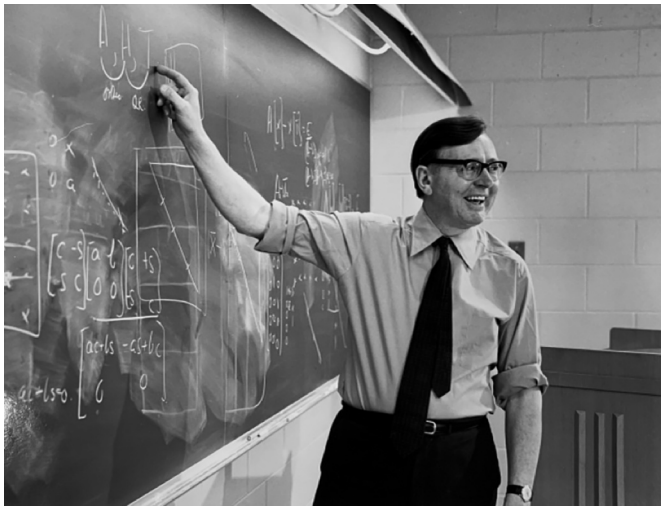

Figure 5. James Wilkinson at the blackboard.

The modern story. Though von Neumann and Goldstine may have been the first to present a more encouraging perspective regarding Gaussian elimination, it wasn't until Wilkinson (Figure 5) came along that a more robust understanding of the computational stability of this algorithm emerged. For instance, a variant of Theorem 1 and the resulting observation that the growth factor $g(A)$ can be used to estimate stability were first realized by Wilkinson [Wil61]. Wilkinson's early work on the subject included the analysis of Gaussian elimination primarily under two popular pivoting strategies:

(1) complete pivoting: permuting rows and columns so that the pivot is always the largest entry in the matrix, i.e., $|A_{11}^{(k)}| = \|A^{(k)}\|_{\max}$, $k = 1, \ldots, n$,
(2) partial pivoting: permuting rows so that the pivot is always the largest entry in the first column, i.e., $|A_{11}^{(k)}| = \|A_{:,1}^{(k)}\|_\infty$, $k = 1, \ldots, n$.

These two pivoting strategies are quite different, a fact reflected by the nature of the questions regarding them. Complete pivoting, famously referred to by von Neumann as "customary" (then called "positioning for size"), is a popular theoretical technique with reasonably good guarantees on stability that, today, is primarily used when high accuracy is desired. Partial pivoting, on the other hand, is a more interesting situation. The need to compare far fewer entries of each iterate $A^{(k)}$ means that partial pivoting is much faster than complete pivoting, and adds little to the overall run time of the Gaussian elimination algorithm. However, the issue here is that this procedure is known to be horribly unstable in the worst-case (see below). Still, Gaussian elimination with partial pivoting remains the most popular method for solving an unstructured linear system numerically: the backslash "\" command in Matlab and Julia, and the linalg.solve() command in Python, all

employ this method (by calling LAPACK routines) when faced with an arbitrary matrix.

Its use in spite of this worst-case instability can be concisely explained as follows: it is simply faster, in a practical sense, than any other known algorithm, and, historically, numerical analysts have believed that the situations where it is unstable are extremely rare. For instance, the QR factorization

$$\underset{A}{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}} = \underset{Q}{\begin{bmatrix} | & | & | \\ q_1 & q_2 & q_3 \\ | & | & | \end{bmatrix}} \underset{R}{\begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}},$$

where A is written as the product of an orthogonal matrix Q and upper triangular matrix R, is probably the most popular alternative to Gaussian elimination. This approach solves a linear system $Ax = b$ by computing $A = QR$ and applying backward substitution to $Rx = Q^T b$. This factorization is mathematically equivalent to applying Gram-Schmidt orthogonalization to the columns of A, though, especially when Q does not need to be formed explicitly, in practice it is often computed by applying simple orthogonal transformations to A to convert it to an upper triangular matrix R. While the QR factorization, computed by, say, Householder reflections, is incredibly stable, it requires roughly $\frac{4}{3}n^3$ operations to compute (with Q only implicitly computed), while a simple counting argument applied to equation (1) implies the LU factorization only requires roughly $\sum_{k=1}^{n} 2k^2 \approx \frac{2}{3}n^3$ operations (k^2 for the vector outer product, k^2 for the subtraction), half the cost compared to QR. While an exact comparison of computational efficiency is far more complicated than simple operation-counting, often involving the movement of data in the computer's memory system, the main takeaway is that Gaussian elimination, paired with partial pivoting, remains the fastest way to compute the solution to an unstructured linear system. In some sense, the use of Gaussian elimination is a trade between guaranteed stability for a factor of two in runtime.

In what remains of this article, I'd like to talk a little about complete pivoting and partial pivoting, in each case starting with the early contributions of Wilkinson and discussing the modern developments, while briefly providing some mathematical insight into the types of techniques involved. It's worth noting that many results do not fall under either complete or partial pivoting, my favorite two being the lower bounds of Higham and Higham for the growth factor of a matrix under any pivoting strategy [HH89] and the estimates of Demmel, Grigori, and Rusciano on the expected log growth of a matrix pre- and post-multiplied by random orthogonal matrices [DGR23]. I encourage the motivated reader to seek these results out, as well as the many other research directions I was unable to mention here due to space constraints.[7] For more details about complete and partial pivoting, Higham's *Accuracy and Stability of Numerical Algorithms* [Hig02] (for complete pivoting) and Trefethen and Bau's *Numerical Linear Algebra* [TB22] (for partial pivoting) are good starting points.

The theory of complete pivoting. In 1961, Wilkinson produced the following bound for a completely pivoted matrix $A \in \mathrm{GL}_n(\mathbb{C})$:

$$g(A) \leq \sqrt{n}\left(2\, 3^{1/2} \dots n^{1/(n-1)}\right)^{1/2} \leq 2n^{\frac{1}{4}\ln n + \frac{1}{2}}. \qquad (3)$$

The proof is incredibly short, requiring one page of mathematics and using only Hadamard's inequality applied to the matrix iterates $A^{(k)} \in \mathbb{C}^{k\times k}$. However, the proof is also somewhat mysterious. I'd like to sketch a slightly more transparent proof through the language of linear programming.

Let $p_k = \|A^{(k)}\|_{\max}$ (recall that, under complete pivoting, this is the magnitude of the pivot of $A^{(k)}$). Hadamard's inequality, that the modulus of the determinant of a matrix is at most the product of the two-norm of its columns, applied to $A^{(k)}$ implies that

$$\prod_{i=1}^{k} p_i = \det(A^{(k)}) \leq k^{k/2} \|A^{(k)}\|_{\max}^k = k^{k/2} p_k^k.$$

The leftmost equality is a result of L having ones and U having the pivots on their respective diagonals. The maximum kth pivot, viewed as a function of k, is nondecreasing, and so the maximum value of p_1/p_n under these constraints provides an upper bound for the maximum growth factor. The transformation $q_k = \ln(p_k)$ produces the linear program:

$$\begin{aligned} \max \quad & q_1 - q_n \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^{k} q_i \leq \frac{k}{2}\ln k + k\, q_k \quad \text{for } k = 1, \dots, n. \end{aligned}$$

We can rewrite this in matrix-vector form as

$$\max \quad c^T x \qquad \text{s.t.} \quad Ax \leq b,$$

where $x = (q_1, \dots, q_n)^T$, $c = (1, 0, \dots, 0, -1)^T$, and $Ax \leq b$ is given by[8]

$$\begin{bmatrix} -1 & & & & \\ 1 & -1 & & & \\ 1 & 1 & -2 & & \\ \vdots & \vdots & \ddots & \ddots & \\ 1 & 1 & \cdots & 1 & -(n-1) \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_n \end{bmatrix} \leq \begin{bmatrix} 0 \\ \frac{2}{2}\ln 2 \\ \frac{3}{2}\ln 3 \\ \vdots \\ \frac{n}{2}\ln n \end{bmatrix},$$

[7] *including rook pivoting, threshold pivoting, the Cholesky factorization, incomplete LU/Cholesky, rank-revealing techniques, communication-avoiding techniques, butterfly matrices for LU, growth for Hadamard matrices, and others...*

[8] *The additional constraint $q_1 \geq 0$ plays no role, as the feasible region for $(q_1, \dots, q_n)$ is shift-independent.*

and holds entrywise. The matrix A has an easily computable inverse, with

$$A^{-1}_{ij} = \begin{cases} -1 & \text{for } i \geq j = 1, \\ -\frac{1}{i-1} & \text{for } i = j > 1, \\ -\frac{1}{j(j-1)} & \text{for } i > j > 1. \end{cases}$$

The quantity $[A^{-1}]^T c$

$$\begin{bmatrix} -1 & -1 & -1 & \cdots & -1 \\ & -1 & -\frac{1}{2} & \cdots & -\frac{1}{2} \\ & & -\frac{1}{2} & & \vdots \\ & & & \ddots & -\frac{1}{(n-2)(n-1)} \\ & & & & -\frac{1}{n-1} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{2} \\ \vdots \\ \frac{1}{(n-2)(n-1)} \\ \frac{1}{n-1} \end{bmatrix}$$

is entrywise nonnegative, implying inequality (3)

$$\begin{aligned} q_1 - q_n &= \left([A^{-1}]^T c\right)^T Ax \\ &\leq \left([A^{-1}]^T c\right)^T b \\ &= \tfrac{1}{2}\left(\ln n + \textstyle\sum_{k=2}^{n} \frac{\ln k}{k-1}\right). \end{aligned}$$

Wilkinson felt that this bound was extremely pessimistic, a viewpoint shared by most numerical analysts today. In the same paper, he remarked that [Wil61]

> In our experience the last pivot has fallen far below the given bound, and no matrix has been encountered in practice for which p_1/p_n was as large as 8.

This sentiment was soon quantified, and a folklore conjecture was formed that the growth factor under complete pivoting of a real $n \times n$ matrix was at most n. A great deal of research went into proving this conjecture for small n. The exact conjecture proved to be false, illustrated by a 13×13 matrix with growth factor 13.02 found numerically by Nick Gould in 1991 [Gou91] and verified in exact arithmetic by Alan Edelman a year later [Ede92]. But still the larger question of how this quantity behaves remains.

Question 2. What is the asymptotic behavior of the maximum growth factor under complete pivoting?

One theoretical consequence of this question is that, if the growth factor is bounded by a polynomial in n, then only $O(\log n)$ bits are required to perform Gaussian elimination accurately.

This question is one that my collaborators and I have made some progress on recently, producing a lower bound of $1.0045n$ for all $n \geq 11$ [EU24] and an upper bound of $n^{0.20781 \ln n + 0.91}$ [BEU25]. The former is the first disproof of the 1960s folklore conjecture for all n larger than a constant. The latter is the first improvement to Wilkinson's 1961 bound, and involves finding further constraints on the pivots of A and solving a much more complicated linear program. However, the gap between the upper and lower bounds for this quantity still remains quite large.

Partial pivoting and a practical problem. In contrast, for partial pivoting, where the rows of A are permuted so that $|A^{(k)}_{11}| \geq |A^{(k)}_{i1}|$ for all i and k, the worst-case behavior is incredibly well understood. Because the pivot $A^{(k)}_{11}$ is the largest entry in the first column, the largest entry of $A^{(k-1)}$ is at most twice that of $A^{(k)}$, leading to an upper bound of 2^{n-1} for the growth factor of an $n \times n$ matrix under partial pivoting. In his 1965 text *The Algebraic Eigenvalue Problem* [Wil65], Wilkinson showed that this bound can be obtained, illustrated by the example matrix

$$A = \begin{bmatrix} 1 & 0 & \cdots & 0 & 1 \\ -1 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & 1 & 0 & 1 \\ -1 & \cdots & -1 & 1 & 1 \\ -1 & \cdots & -1 & -1 & 1 \end{bmatrix}. \tag{4}$$

After one step of Gaussian elimination, the resulting matrix looks nearly identical in structure to the original, except the last column has doubled in size:

$$A_{2:n,2:n} - (-\mathbf{1})\vec{e}^{\,T}_{n-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 2 \\ -1 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & 1 & 0 & 2 \\ -1 & \cdots & -1 & 1 & 2 \\ -1 & \cdots & -1 & -1 & 2 \end{bmatrix},$$

where $\mathbf{1} = (1, \ldots, 1)^T$. This doubling occurs at every step, resulting in the last entry of U equaling 2^{n-1}. This exponentially large growth factor can lead to catastrophic errors, even for well-conditioned matrices.

To see this in practice, let's consider the solution to a 100×100 linear system $Ax = b$, where A is the example matrix in equation (4), x is a Gaussian vector, and, of course, b equals A times x. This setup allows us to explicitly compute the numerical error, as we already know x exactly. Using the Julia programming language, we first define the Wilkinson matrix from equation (4):

```
julia> wilk(n) = [ i>j ? -1 : (i==j ||
         j==n) ? 1 : 0 for i=1:n, j=1:n];
```

Next, we initialize A, randomly select x, and compute $b = Ax$:

```
julia> A = wilk(100);
       x = randn(100);
       b = A*x;
```

Solving for x numerically using the built-in "\" command, which employs Gaussian elimination with partial pivoting, we're in for quite a surprise:

```
julia> norm(x-A\b)/norm(x)
  1.8623380318378875
```

The computed solution (typically accurate to $\approx 10^{-13}$) has relative error greater than one (e.g., zero is a better guess)! To really stress the inaccuracy of the computed solution, we can take a look at the last ten digits of x and our computed $A\backslash b$:

```
julia> [x A\b][91:100,:]
  10×2 MatrixFloat64:
   0.727419    0.0
  -0.383269    0.0
  -2.3035      0.0
   0.134562    0.0
   2.42309     0.0
   1.46416     0.0
   0.358267    0.0
  -0.375875    0.0
   0.817008    0.0
   0.0752026   0.0752026
```

This is not caused by the condition number:

```
julia> cond(A)
  44.80225124630292
```

as linear condition numbers are quite acceptable for most applied problems. No warning or error message is given. The same phenomenon occurs in the majority of programming languages (e.g., "\" in Matlab, linalg.solve() in Python, etc.).

Many researchers suspect that the situations where partial pivoting fails are rare. In *Matrix Computations*, Golub and Van Loan state that [GVL13, p. 131]

> Although there is still more to understand about ρ [the growth factor], the consensus is that serious element growth in Gaussian elimination with partial pivoting is *extremely* rare. *The method can be used with confidence*.

Of course, even rare instances become important to understand as the number and variety of computations increases exponentially. For partial pivoting, the majority of modern research is devoted to either quantifying how rare this worst-case behavior is or illustrating how this worst-case behavior can be avoided altogether. In 2012, Nick Trefethen, the then-president of SIAM, described "two of the biggest unsolved problems in numerical analysis" [Tre12]. The first, unsurprisingly, was fast matrix multiplication; the second was "why does Gaussian elimination work?" Inspired by experiments, he conjectured that the growth factor of a Gaussian matrix under partial pivoting scales like $\sqrt{n}$ and offered $1000 for a proof:

Conjecture 3. *Let A be an $n \times n$ Gaussian matrix, i.e., A_{ij} iid $N(0,1)$. Then, for any $\alpha, M > 0$, the probability that the growth factor of A under partial pivoting is larger than $n^{1/2+\alpha}$ is at most n^{-M} for all n sufficiently large.*

While some asymptotic properties require large values of n to emerge, here the $\sqrt{n}$ behavior of the growth factor is abundantly clear, even for small values of n (Figure 6). However, proving this exactly seems difficult, for reasons we will soon discuss.

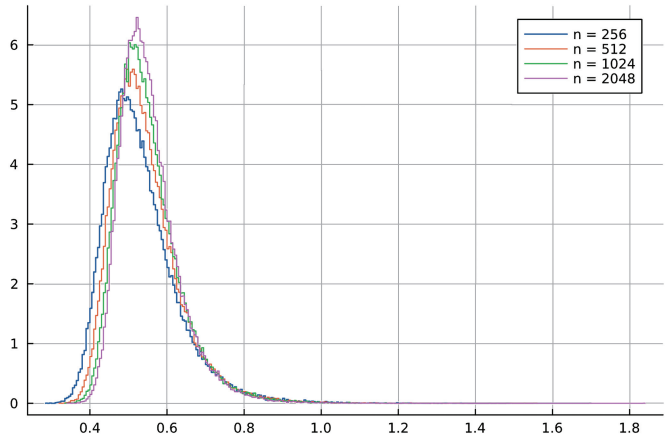

Figure 6. Histograms of 100,000 samples of the growth factor under partial pivoting divided by $\sqrt{n}$ of $\mathrm{randn}(n,n)$ for $n = 2^k$, $k = 8, \ldots, 11$. The largest growth factor encountered here was less than $2\sqrt{n}$.

Another approach for analyzing partial pivoting is through *smoothed analysis*, a technique popularized by Spielman and Teng that strikes a balance between average-case and worst-case analysis. A smoothed analysis of an algorithm is a study of its performance for a mild random perturbation of an arbitrary input. For example, even though Wilkinson's matrix (equation (4)) is a highly unstable instance, upon mild perturbation (even without pivoting), the growth factor decreases significantly (Figure 7). In some sense, smoothed analysis is a generalization of average-case analysis, with the added benefit of being not only descriptive, but prescriptive. In particular, rather than speculate about the rarity of unstable instances in practice, smoothed analysis offers a concrete solution to avoid these instances altogether, in a true probabilistic sense, through the introduction of a modest amount of noise.

In 2006, Sankar, Spielman, and Teng provided a smoothed analysis of Gaussian elimination without pivoting [SST06]. Here's one sample result: for an arbitrary $n \times n$ matrix A with $\|A\|_2 \leq 1$, and a $n \times n$ Gaussian matrix B, the LU factorization $LU = A + \epsilon B$, $\epsilon \leq 1$, satisfies

$$\mathbb{P}\left[\frac{\|U\|_\infty}{\|A+\epsilon B\|_\infty} > 1 + t\right] < \frac{1}{\sqrt{2\pi}}\frac{n(n+1)}{t\epsilon}. \qquad (5)$$

I'd like to give some intuition about how a result like this, or, say, one about a Gaussian matrix, is often proven. By equation (2), the entries of U are elements of a Schur complement involving the leading submatrices of A, and the only way the elements of U can become much larger

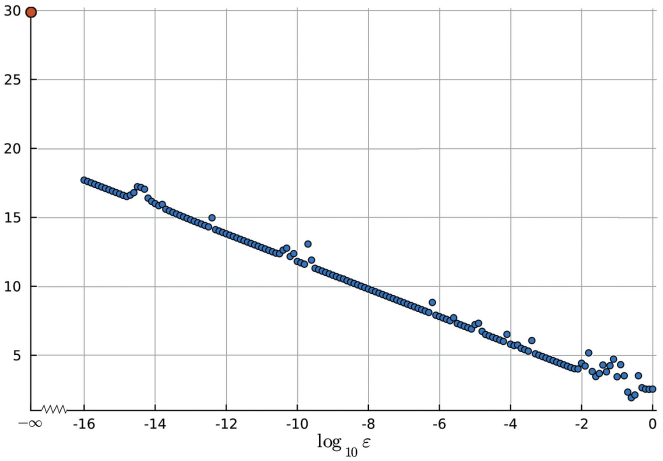

Figure 7. A log-log plot of $g(A+\epsilon B)$ (the growth factor without pivoting) with respect to ϵ for a 100×100 Wilkinson matrix A (equation (4)) and a fixed Gaussian matrix B. Setting $\epsilon = 10^{-16}$ (approximately the unit round-off $\mathbf{u}$ in double precision) already drops the growth from approximately 10^{30} to 10^{18}.

than those of A is if a leading submatrix of A has a very small singular value. In practice, growth factor estimates for these types of problems essentially reduce to singular value-type estimates, or, similarly, bounding the application of the inverse of a leading submatrix to a given vector. For example, inequality (5) essentially relies on the following property of a random Gaussian perturbation:

Lemma 4 ([SST06]). *Let $A \in \mathbb{R}^{n\times n}$ and B be a Gaussian matrix. Then, for any fixed unit vector $x \in \mathbb{R}^n$,*

$$\mathbb{P}[\|(A+\epsilon B)^{-1}x\|_2 > t] < \sqrt{\frac{2}{\pi}}\frac{1}{t\epsilon}.$$

While I won't provide a proof of this, I can give the basic idea in a few lines. Because Gaussian matrices are distributionally invariant under orthogonal transformations, we can assume $x = (1, 0, \dots, 0)^T$ and simply bound the size of the first column of $(A+\epsilon B)^{-1}$. The size of this column is exactly the inverse of the inner product between the first row of $A + \epsilon B$ and a unit vector from the orthogonal complement of the remaining $n-1$ rows, reducing the problem to one involving the inner product between a fixed unit vector and a Gaussian one, which is well studied.

Overall, the average-case and smoothed analysis without pivoting is fairly well understood. However, making analogous statements for partial pivoting has proven much more difficult. Partial pivoting can be viewed as a greedy determinant maximization where the kth row is chosen to maximize the leading $k \times k$ minor. Intuitively, this reordering should significantly improve the situation, as larger leading minors seem beneficial. This intuition is supported by practice, as partial pivoting is usually, though certainly not always, far superior to no pivoting. However, this greedy procedure makes mathematical analysis significantly more difficult, as it introduces dependence between elements of the random perturbation.

For instance, Sankar, in his PhD thesis [San04], performed a smoothed analysis of partial pivoting, but, due to the mathematical complications introduced by pivoting, was only able to prove a quasipolynomial bound for growth. Recently, Huang and Tikhomirov managed to prove that the growth factor under partial pivoting of a Gaussian matrix is polynomial [HT24]. One key idea was a quantification of how "Gaussian" the remaining $n-k$ rows of a Gaussian matrix are after k rows have already been chosen during partial pivoting, e.g., understanding how much randomness persists in spite of the pivoting procedure. This result is still quite far from the expected $\sqrt{n}$ behavior, and the arguments presented do not appear to directly translate to the smoothed analysis setting.

There is still quite a gap between what we observe (Figure 6) and what we can prove mathematically, though. The problems of producing a smoothed analysis of partial pivoting with a polynomial bound, or better yet, one at least as strong as inequality (5), and proving Trefethen's $1000 conjecture both remain completely open.

ACKNOWLEDGMENTS. The author thanks Louisa Thomas and two anonymous referees for significantly improving the style of presentation.

All experiments were performed in the Julia programming language, on a Mac Studio with an Apple M2 Max chip and 32GB of unified memory. Figures 6 and 7 were created using the built-in lu() command in double and quadruple precision, respectively.

References

[BEU25] Ankit Bisain, Alan Edelman, and John Urschel, *A new upper bound for the growth factor in Gaussian elimination with complete pivoting*, Bulletin of the London Mathematical Society (2025).

[BMvN46] Valentine Bargmann, Deane Montgomery, and John von Neumann, *Solution of linear systems of high order*, US Department of Commerce, Office of Technical Services, 1946.

[DGR23] James Demmel, Laura Grigori, and Alexander Rusciano, *An improved analysis and unified perspective on deterministic and randomized low-rank matrix approximation*, SIAM J. Matrix Anal. Appl. **44** (2023), no. 2, 559–591, DOI 10.1137/21M1391316. MR4586593

[Ede92] Alan Edelman, *The complete pivoting conjecture for Gaussian elimination is false* (1992).

[EU24] Alan Edelman and John Urschel, *Some new results on the maximum growth factor in Gaussian elimination*, SIAM J. Matrix Anal. Appl. **45** (2024), no. 2, 967–991, DOI 10.1137/23M1571903. MR4737248

[Gol93] Herman H. Goldstine, *The computer from Pascal to von Neumann*, Princeton University Press, Princeton, NJ, 1993. Reprint of the 1972 original. MR1271142

[Gou91] Nick Gould, *On growth in Gaussian elimination with complete pivoting*, SIAM J. Matrix Anal. Appl. **12** (1991), no. 2, 354–361, DOI 10.1137/0612025. MR1089164
[GVL13] Gene H. Golub and Charles F. Van Loan, *Matrix computations*, 4th ed., Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 2013. MR3024913
[Har11] Roger Hart, *The Chinese roots of linear algebra*, Johns Hopkins University Press, Baltimore, MD, 2011. MR2731645
[HH89] Nicholas J. Higham and Desmond J. Higham, *Large growth factors in Gaussian elimination with pivoting*, SIAM J. Matrix Anal. Appl. **10** (1989), no. 2, 155–164, DOI 10.1137/0610012. MR988627
[Hig02] Nicholas J. Higham, *Accuracy and stability of numerical algorithms*, 2nd ed., Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2002, DOI 10.1137/1.9780898718027. MR1927606
[Hot43] Harold Hotelling, *Some new methods in matrix calculation*, Ann. Math. Statistics **14** (1943), 1–34, DOI 10.1214/aoms/1177731489. MR7851
[HT24] Han Huang and Konstantin Tikhomirov, *Average-case analysis of the Gaussian elimination with partial pivoting*, Probab. Theory Related Fields **189** (2024), no. 1-2, 501–567, DOI 10.1007/s00440-024-01276-2. MR4750563
[San04] Arvind Sankar, *Smoothed analysis of Gaussian elimination*, ProQuest LLC, Ann Arbor, MI, 2004. Thesis (Ph.D.)–Massachusetts Institute of Technology. MR2717147
[SST06] Arvind Sankar, Daniel A. Spielman, and Shang-Hua Teng, *Smoothed analysis of the condition numbers and growth factors of matrices*, SIAM J. Matrix Anal. Appl. **28** (2006), no. 2, 446–476, DOI 10.1137/S0895479803436202. MR2255338
[TB22] Lloyd N. Trefethen and David Bau III, *Numerical linear algebra*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2022. 25th anniversary edition [of 1444820]; With a foreword by James G. Nagy. MR4713493
[Tre12] Lloyd N. Trefethen, *The smart money's on numerical analysts*, SIAM News **45** (2012), no. 9, 1–5.
[vNG47] John von Neumann and H. H. Goldstine, *Numerical inverting of matrices of high order*, Bull. Amer. Math. Soc. **53** (1947), 1021–1099, DOI 10.1090/S0002-9904-1947-08909-6. MR24235
[Wil61] J. H. Wilkinson, *Error analysis of direct methods of matrix inversion*, J. Assoc. Comput. Mach. **8** (1961), 281–330, DOI 10.1145/321075.321076. MR176602
[Wil65] J. H. Wilkinson, *The algebraic eigenvalue problem*, Clarendon Press, Oxford, 1965. MR184422

John Urschel

Credits

Figure 1 is in the public domain, `https://commons.wikimedia.org/w/index.php?curid=66240126`.
Figure 2, Figure 6, and Figure 7 are courtesy of John Urschel.
Figure 3 is courtesy of Fresheneesz at the English Wikipedia project, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=3357169`.
Figure 4 is courtesy of Nicholas J. Higham.
Figure 5 is courtesy of Jack Dongarra.
Photo of John Urschel is courtesy of Christopher Harting.