STAT 309: MATHEMATICAL COMPUTATIONS I
FALL 2023
LECTURE 9

1. BACKWARD STABILITY

- the type of analysis we did in the previous lecture is very useful and is called *backward error analysis*
- more generally, we regard our *problem* as a function $f : X \to Y$ that takes input $x \in X$ (elements in the domain of f) to output $y \in Y$ (elements in the codomain of f)
- strictly speaking, this is only correct if we have a well-posed problem, i.e., one with guaranteed existence and uniqueness of solution (every element in the domain gets mapped to exactly one image in the codomain)
- for example, the problem of LU factorization is $f : \mathbb{R}^{n\times n} \to \mathfrak{S}_n \times \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n}$, $f(A) = (L, U)$ where[1] $A = \Pi^{\mathsf{T}} LU$
- for example, the problem of solving linear systems is $f : \mathrm{GL}(n) \times \mathbb{R}^n \to \mathbb{R}^n$, $f(A, \mathbf{b}) = A^{-1}\mathbf{b}$
- given an input $x \in X$, an algorithm for producing an output $y = f(x)$ is subjected to rounding errors and would instead produces a computed output $\widehat{y}$
- the key in backward error analysis is to assume that

  the computed solution $\widehat{y}$ is the exact solution of a perturbed input $x + \Delta x$

  i.e., $\widehat{y} = f(x + \Delta x)$
- the algorithm is said to be *backward stable* if for any $x \in X$, the computed $\widehat{y}$ satisfies
$$\widehat{y} = f(x + \Delta x), \quad |\Delta x| \le \delta |x|$$
  for some 'small' δ
- Δx is called the *backward error* while $\widehat{y} - y$ is called the *forward error*
- $|\cdot|$ is some measure of the 'size' of x, usually a norm
- see Figure 1 for a pictorial depiction of the above discussion

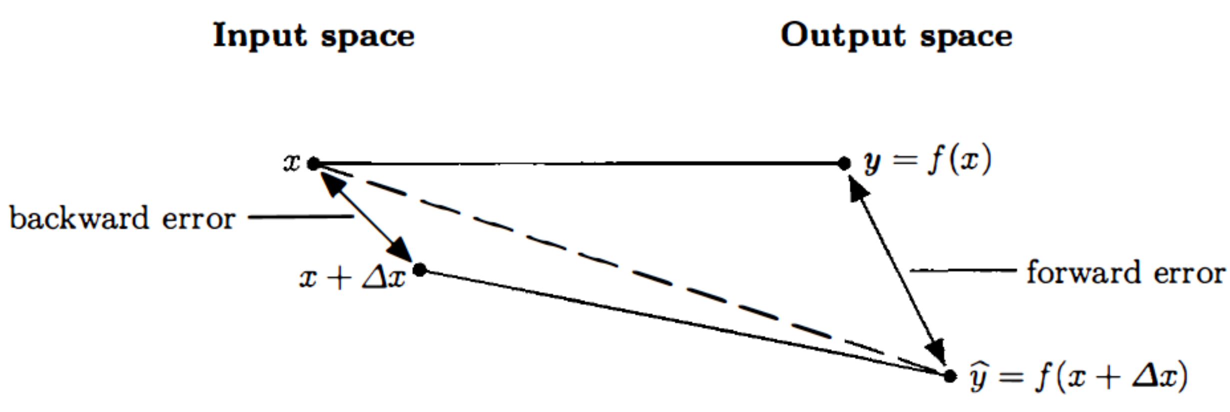

FIGURE 1. solid line = exact; dotted-line = computed; taken from N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd Ed, SIAM, 2002

Date: October 30, 2023, version 1.0. Comments, bug reports: `lekheng@uchicago.edu`.

[1] $\mathfrak{S}_n$ is the symmetric group, i.e., set of all permutations of n objects

- for example, take the problem of finding singular value decomposition
$$f : \mathbb{R}^{m\times n} \to \mathrm{O}(m) \times \mathbb{R}^{\min(m,n)} \times \mathrm{O}(n), \quad A \mapsto (U, \boldsymbol{\sigma}, V)$$
where $U\Sigma V^{\mathsf{T}} = A$ and $\Sigma = \operatorname{diag}(\boldsymbol{\sigma})$
- suppose we have an algorithm $\widehat{f}$ for computing f, then the output of $\widehat{f}$ on input A is
$$\widehat{f}(A) = (\widehat{U}, \widehat{\boldsymbol{\sigma}}, \widehat{V})$$
- say we use the Frobenius norm to measure error, then the (absolute) forward errors are
$$\|U - \widehat{U}\|_{\mathsf{F}}, \quad \|\Sigma - \widehat{\Sigma}\|_{\mathsf{F}}, \quad \|V - \widehat{V}\|_{\mathsf{F}}$$
we can't compute any of these since we do not have U, Σ, V — we only have $\widehat{U}, \widehat{\Sigma}, \widehat{V}$
- the (absolute) backward error is
$$\|A - \widehat{U}\widehat{\Sigma}\widehat{V}^{\mathsf{T}}\|_{\mathsf{F}}$$
which can be readily computed
- for another example, take the problem of solving linear system $A\mathbf{x} = \mathbf{b}$ with some fixed $A \in \mathrm{GL}(n)$, i.e.,
$$f : \mathbb{R}^n \to \mathbb{R}^n, \quad \mathbf{b} \mapsto \mathbf{x} = A^{-1}\mathbf{b}$$
- suppose we have an algorithm $\widehat{f}$ for computing f, then the output of $\widehat{f}$ on input $\mathbf{b}$ is
$$\widehat{f}(\mathbf{b}) = \widehat{\mathbf{x}}$$
- we can't compute (absolute) forward error $\|\widehat{\mathbf{x}} - \mathbf{x}\|$ since it requires the exact solution $\mathbf{x}$
- but the (absolute) backward error $\|A\widehat{\mathbf{x}} - \mathbf{b}\|$, also called the *residual*, can be readily computed from the computed solution $\widehat{\mathbf{x}}$

2. Numerical stability

- the above notion of backward stability above is too restrictive to in most instances
- one reason is that the computed $\widehat{y}$ may not even be in the range of f, i.e., $\widehat{y} \neq f(x + \Delta x)$ for any choice of Δx
- another reason is that even if $\widehat{y} = f(x + \Delta x)$ for some Δx, it may be too difficult to find a reasonable estimate for δ so that $|\Delta x| \le \delta |x|$
- so we use a more convenient notion called *mixed forward-backward stability* when we talk about numerical stability
- an algorithm is said to be *numerically stable* if for any $x \in X$, the computed $\widehat{y}$ satisfies
$$\widehat{y} + \Delta y = f(x + \Delta x), \quad |\Delta x| \le \delta |x|, \quad |\Delta y| \le \varepsilon |y| \tag{2.1}$$
for some 'small' δ and ε, usually on the order of $\varepsilon_{\text{machine}}$
- the way to interpret (2.1) is: "$\widehat{y}$ is almost the right answer for almost the right data"
- see Figure 2 for a pictorial depiction of the above discussion

3. Conditioning and stability

- *conditioning* is a property of a problem whereas *stability* is a property of an algorithm
- the sources of inaccuracy in a computed result could either be due to conditioning or stability or both
- for example the algorithm that computes vector 2-norm via
$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}$$

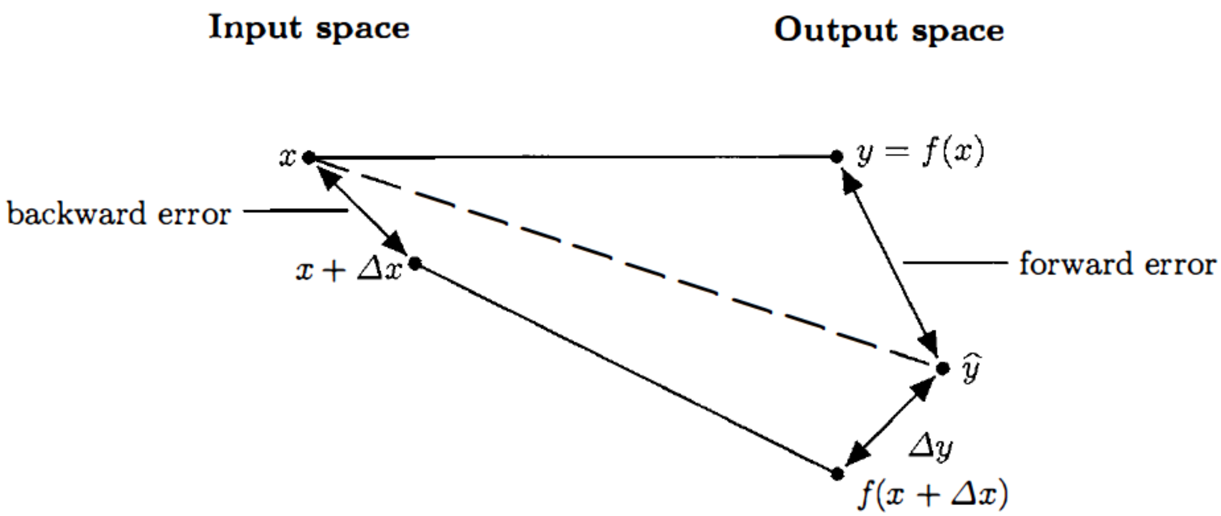

FIGURE 2. solid line = exact; dotted-line = computed; taken from N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd Ed, SIAM, 2002

is less stable than the one that computes it by

$$\widehat{\mathbf{x}} = \begin{cases} \mathbf{x}/\|\mathbf{x}\|_\infty & \text{if } \|\mathbf{x}\|_\infty \neq 0, \\ \mathbf{0} & \text{if } \|\mathbf{x}\|_\infty = 0, \end{cases} \qquad \|\mathbf{x}\|_2 = \|\mathbf{x}\|_\infty \|\widehat{\mathbf{x}}\|_2$$

- cancellation error on the other hand is a problem caused by the conditioning of substraction — when the inputs are nearby, the problem is ill-conditioned
- if $x = a - b$ and $\widehat{x} = \widehat{a} - \widehat{b}$ with $\widehat{a} = a(1 + \Delta a)$ and $\widehat{b} = b(1 + \Delta b)$ where Δa and Δb are relative errors, then
$$\frac{|\widehat{x} - x|}{|x|} = \left| \frac{-a\Delta a + b\Delta b}{a - b} \right| \leq \max(|\Delta a|, |\Delta b|) \frac{|a| + |b|}{|a - b|}$$
and the condition number $(|a| + |b|)/|a - b|$ is large when $a \approx b$
- the backward error analysis we performed may seem a little weird the first time you see it: why don't we assume that the error is in the input and then see how big it becomes in the output — this is called forward error analysis
- forward error analysis is in general much harder than backward error analysis
- fortunately we have a rule-of-thumb that
$$\textit{forward error} \lesssim \textit{condition number} \times \textit{backward error}$$
where '$\lesssim$' means 'roughly bounded by'
- for example, in Homework **2**, Problem **6**(c), we saw that for solving $A\mathbf{x} = \mathbf{b}$ (with no error in $\mathbf{b}$), the forward error $\|\Delta\mathbf{x}\|/\|\mathbf{x}\|$ is related to the backward error $\|\Delta A\|/\|A\|$ via
$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(A)\dfrac{\|\Delta A\|}{\|A\|}}{1 - \kappa(A)\dfrac{\|\Delta A\|}{\|A\|}}$$
- this relation is an example of '$\lesssim$', if we use the expansion $x/(1 - x) \approx x$, we get a simplification
$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \lesssim \kappa(A)\frac{\|\Delta A\|}{\|A\|}$$

- later we will see that if we use Gaussian elimination to solve a linear system in IEEE floating point arithmetic, i.e., all errors ΔA are due to rounding errors, then
$$\frac{\|\Delta A\|_\infty}{\|A\|_\infty} \le n(n+1)\gamma_n \varepsilon_{\text{machine}}$$
for some constant γ_n depending on the pivoting scheme we choose
- let us look at another simple example, matrix multiplication $X \mapsto XA$ by a fixed matrix A
- for simplicity, let us assume that
  - we know the input X precisely
  - all errors arise from rounding in floating point arithmetic and storage
- we don't have XA, only $\mathrm{fl}(XA)$, which differs from XA by an error term E
$$\mathrm{fl}(XA) = XA + E$$
- we will do a backward error analysis again, i.e., we want to find the smallest perturbation ΔA in A so that $XA + E$ is the *exact* answer had $A + \Delta A$ been the input
- we will measure how good our method is by asking what is the relative error in the input
$$\frac{\|\Delta A\|_2}{\|A\|_2} \tag{3.1}$$
required so that the relative error of the output is
$$\frac{\|E\|_2}{\|XA\|_2} \le \varepsilon \tag{3.2}$$
for some $\varepsilon > 0$
- the ratio in (3.1) is the *relative backward error*, the ratio in (3.2) is the *relative forward error*
- in this case, it is trivial to derive the relative backward error: by assumption $XA + E$ is the *exact* answer of multiplying X to $A + \Delta A$, so
$$XA + E = X(A + \Delta A)$$
and so
$$\Delta A = X^{-1}E$$
- from (3.2), we get $\|E\|_2 \le \varepsilon\|XA\|_2 \le \varepsilon\|X\|_2\|A\|_2$ and so
$$\|\Delta A\|_2 \le \|X^{-1}\|_2\|E\|_2 \le \varepsilon\kappa_2(X)\|A\|_2$$
and so the relative backward error is
$$\frac{\|\Delta A\|_2}{\|A\|_2} \le \varepsilon\kappa_2(X) \tag{3.3}$$
- recap of backward error analysis
  - we assume that the error E in the final computed output comes from the *exact* solution of a perturbed problem $A + \Delta A$
  - we start by assuming that the relative error in the output is ε, i.e., (3.2)
  - then we try to find how far away (i.e., ΔA) the input must be from the given one (i.e., A) in order to produce such an error ε in the output, i.e., (3.3), when everything is done without error

4. QR and complete orthogonal factorization

- poor man's SVD
- can solve many problems on the SVD list using either of these factorizations
- but they are much cheaper to compute — there are direct algorithms for computing QR and complete orthogonal factorization in a finite number of arithmetic steps
- recall that SVD is spectral in nature — only iterative algorithms in general by Galois–Abel, although for any fixed precision (fixed number of decimal places), we can compute SVD in finitely many steps
- there are several versions of QR factorization
- version 1: for any $A \in \mathbb{C}^{m\times n}$ with $n \le m$, there exist a unitary matrix $Q \in \mathbb{C}^{m\times m}$ (i.e., $Q^*Q = QQ^* = I_n$) and an upper-triangular matrix $R \in \mathbb{C}^{m\times n}$ (i.e., $r_{ij} = 0$ whenver $i > j$) such that
$$A = QR = Q\begin{bmatrix} R_1 \\ 0 \end{bmatrix} \tag{4.1}$$
  - $R_1 \in \mathbb{C}^{n\times n}$ is an upper-triangular square matrix in general
  - if A has full column rank, i.e., $\operatorname{rank}(A) = n$, then R_1 is nonsingular
  - this is called the *full* QR factorization of A
- version 2: for any $A \in \mathbb{C}^{m\times n}$ with $n \le m$, there exist an orthonormal matrix $Q_1 \in \mathbb{C}^{m\times n}$ (i.e., $Q_1^*Q_1 = I_n$ but $Q_1Q_1^* \ne I_m$ unless $m = n$) and an upper-triangular square matrix $R_1 \in \mathbb{C}^{n\times n}$ such that
$$A = Q_1R_1 \tag{4.2}$$
  - R_1 here is in fact the same R_1 as in (4.1)
  - Q_1 is the first n columns of Q in (4.1), i.e., $Q = [Q_1, Q_2]$ where $Q_2 \in \mathbb{C}^{m\times(m-n)}$ is the last $m - n$ columns of Q
  - in fact we obtain (4.2) from (4.1) by simply multiplying out
$$A = QR = [Q_1, Q_2]\begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1R_1 + Q_2 0 = Q_1R_1$$
  - as before, if A has full column rank, i.e., $\operatorname{rank}(A) = n$, then R_1 is nonsingular
  - this is called the *reduced* QR factorization of A
- version 3: for any $A \in \mathbb{C}^{m\times n}$ with $\operatorname{rank}(A) = r$, there exist a permutation matrix $\Pi \in \mathbb{C}^{n\times n}$, a unitary matrix $Q \in \mathbb{C}^{m\times m}$, and a nonsingular, upper-triangular square matrix $R_1 \in \mathbb{C}^{r\times r}$ such that
$$A\Pi = Q\begin{bmatrix} R_1 & S \\ 0 & 0 \end{bmatrix} \tag{4.3}$$
  - $S \in \mathbb{C}^{r\times(n-r)}$ is just some matrix with no special properties
  - this is called the *rank-retaining* QR decomposition of A form
  - we may also write (4.3) as
$$A = QR\Pi^{\mathsf{T}} = Q\begin{bmatrix} R_1 & S \\ 0 & 0 \end{bmatrix}\Pi^{\mathsf{T}} \tag{4.4}$$
- version 4: for any $A \in \mathbb{C}^{m\times n}$ with $\operatorname{rank}(A) = r$, there exist a unitary matrix $Q \in \mathbb{C}^{m\times m}$, a unitary matrix $U \in \mathbb{C}^{n\times n}$, and a nonsingular, lower-triangular square matrix $L \in \mathbb{C}^{r\times r}$ such that
$$A = Q\begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix}U^* \tag{4.5}$$
  - this is called the *complete orthogonal* factorization of A

- it can be obtained from a full QR factorization of $\begin{bmatrix} R_1^* \\ S^* \end{bmatrix} \in \mathbb{C}^{m\times r}$, which has full column rank,
$$\begin{bmatrix} R_1^* \\ S^* \end{bmatrix} = Z \begin{bmatrix} R_2 \\ 0 \end{bmatrix} \tag{4.6}$$
where $Z \in \mathbb{C}^{m\times m}$ is unitary and $R_2 \in \mathbb{C}^{r\times r}$ is nonsingular, upper-triangular square matrix
- observe from (4.4) and (4.6) that
$$A = Q \begin{bmatrix} R_1 & S \\ 0 & 0 \end{bmatrix} \Pi^{\mathsf{T}} = Q \begin{bmatrix} R_2^* & 0 \\ 0 & 0 \end{bmatrix} Z^* \Pi^{\mathsf{T}} = Q \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} U^*$$
where we set $L = R_2^*$ and $U = \Pi Z$.

- note that for a matrix that is not of full column rank, a QR decomposition would necessarily mean either versions 3 or 4
- there are yet other variants of QR factorizations that can be obtained using essentially the same algorithms (Givens and Householder QR):
$$A = QR, \quad A = LQ, \quad A = RQ, \quad A = QL$$
where Q is unitary, R is upper triangular, and L is lower triangular
- using such variants, we could for instance make the lower triangular matrix L in (4.5) an upper-triangular matrix instead
- the QR factorization is sometimes regarded as a generalization of the polar form of a complex number $a \in \mathbb{C}$,
$$a = re^{i\theta}$$
to matrices, we will see later that we may always choose our R so that $r_{ii} \geq 0$

5. Aside: permutation matrices

- the permutation matrix Π in (4.3) comes from performing *column pivoting* in the algorithm
- recall that a permutation matrix is a simply the identity matrix with the rows and columns permuted, e.g.
$$\Pi = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{5.1}$$
- multiplying a matrix $A \in \mathbb{C}^{m\times n}$ by an $n \times n$ permutation matrix on the right, i.e., $A\Pi$, has the effect of permuting the *columns* of A according to precisely the way the columns of Π are permuted from the identity, e.g.
$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} c & a & b \\ f & d & e \\ i & g & h \end{bmatrix}$$
- multiplying a matrix $A \in \mathbb{C}^{m\times n}$ by an $m \times m$ permutation matrix on the left, i.e., ΠA, has the effect of permuting the *rows* of A according to precisely the way the rows of Π are permuted from the identity, e.g.
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} d & e & f \\ g & h & i \\ a & b & c \end{bmatrix}$$

- multiplying a square matrix $A \in \mathbb{C}^{n\times n}$ by an $n \times n$ permutation matrix on the left and its transpose on the right, i.e., $\Pi A \Pi^{\mathsf{T}}$, has the effect of permuting the *diagonal* of A — entries on the diagonal stays on the diagonal and entries off the diagonal stays off diagonal, e.g.
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}} = \begin{bmatrix} e & f & d \\ h & i & g \\ b & c & a \end{bmatrix}$$
note that a, e, i stays on the diagonal as expected
- permutation matrices are always orthogonal (also unitary since it has real entries), i.e.
$$\Pi^{\mathsf{T}}\Pi = \Pi\Pi^{\mathsf{T}} = I$$
or $\Pi^{-1} = \Pi^{\mathsf{T}} = \Pi^*$
- we don't store permutation matrices as matrices of floating point numbers, we store just the permutation, e.g. (5.1) can be stored as $3 \mapsto 1 \mapsto 2 \mapsto 3$ since it takes column 3 to column 1, column 1 to column 2, column 2 to column 3

6. Existence and uniqueness of QR

- if $A \in \mathbb{C}^{m\times n}$ has full column rank, i.e., $\operatorname{rank}(A) = n \le m$, then we will show existence and (some kind of) uniqueness of its reduced QR factorization
- uniqueness is easy if $m = n$
  - suppose
  $$A = Q_1R_1 = Q_2R_2$$
  for $Q_1, Q_2 \in \mathbb{C}^{n\times n}$ are unitary and $R_1, R_2 \in \mathbb{C}^{n\times n}$ are nonsingular
  - then
  $$Q_2^*Q_1 = R_2R_1^{-1}$$
  - note that the left-hand side is unitary and right hand side is upper-triangular
  - the only matrix that is both unitary and upper-triangular is a diagonal matrix of the form
  $$D = \operatorname{diag}(e^{i\theta_1}, \dots, e^{i\theta_n})$$
  - so we get
  $$Q_2 = Q_1D^*, \qquad R_2 = DR_1$$
  - QR factorization is unique up to such unimodular scaling
- more generally, we could also get uniqueness without requiring $m = n$ this follows from Gram–Schmidt, which we could also use to establish existence