Outline

1. Student Learning Targets, and Objectives
   - SLOs: Floating Point Arithmetic & Stability

2. Stability
   - Recap: Floating Point; Stability Definitions
   - The Road Ahead

3. Stability of Floating Point Arithmetic
   - Basic Operations
   - Inner Product; Outer Product
   - $x + C$

4. Examples
   - $\sin$ and $\cos$
   - Matrix Eigenvalues

5. Accuracy
   - Accuracy of a Backward Stable Algorithm
   - Stability + Conditioning $\sim$ Quality
   - Backward Error Analysis
   - Looking Forward: Application of Backward Error Analysis

Peter Blomgren ⟨blomgren@sdsu.edu⟩ 11. Stability... a Closer Look
Target  Backward Stability of Basic Floating Point Arithmetic
  Objective  Know the procedure for showing that $\oplus$, $\ominus$, $\otimes$, and $\oslash$ are backward stable.

Objective

Target  ...
  Objective
  Objective
  Objective
Axiom (Floating Point Representation)

∀x ∈ ℝ, there exists ε with |ε| ≤ ε_{mach}, such that \( \text{fl}(x) = x(1 + \varepsilon) \).

Axiom (The Fundamental Axiom of Floating Point Arithmetic)

For all \( x, y \in \mathbb{F}_n \) (where \( \mathbb{F}_n \) is the set of \( n \)-bit floating point numbers), there exists ε with |ε| ≤ ε_{mach}(\( \mathbb{F}_n \)), such that

\[
\begin{align*}
  x \oplus y &= (x + y)(1 + \varepsilon), \\
  x \ominus y &= (x - y)(1 + \varepsilon), \\
  x \otimes y &= (x \cdot y)(1 + \varepsilon), \\
  x \oslash y &= (x / y)(1 + \varepsilon)
\end{align*}
\]

Above, \( \text{fl} : \mathbb{R} \mapsto \mathbb{F}_n \) is the “function” which takes a real number and produces its \( n \)-bit floating point representation.
Definition (Stable Algorithm)

We say that \( \tilde{f} \) is a **stable algorithm** if \( \forall \tilde{x} \in X \)

\[
\frac{\| \tilde{f}(\tilde{x}) - f(\tilde{x}) \|}{\| f(\tilde{x}) \|} = O(\varepsilon_{\text{mach}})
\]

for some \( \tilde{x} \) with

\[
\frac{\| \tilde{x} - \tilde{x}' \|}{\| \tilde{x}' \|} = O(\varepsilon_{\text{mach}})
\]

“A stable algorithm gives approximately the right answer, to approximately the right question.”

Peter Blomgren (blomgren@sdsu.edu)
Definition (Backward Stable Algorithm)

An algorithm \( \tilde{f} \) is **backward stable** if \( \forall \tilde{x} \in X \)

\[
\tilde{f}(\tilde{x}) = f(\tilde{x})
\]

for some \( \tilde{x} \) with

\[
\frac{\|\tilde{x} - \tilde{x}'\|}{\|\tilde{x}'\|} = \mathcal{O}(\varepsilon_{\text{mach}})
\]

"A backward stable algorithm gives exactly the right answer, to approximately the right question."

Peter Blomgren (blomgren@sdsu.edu)
Stability: The Road Ahead

- **Backward Error Analysis** — linking conditioning *(which is a property of the underlying mathematical problem)* and stability *(which is a property of the algorithm)*.
- Detailed Stability Analysis (backward error analysis) of Householder Triangularization.
We start off by showing that our algorithmic building blocks — the floating point operations $\oplus$, $\ominus$, $\otimes$, and $\oslash$ are backward stable.

We look at subtraction, which may be the biggest cause for concern due to cancellation errors. For $\vec{x} = [x_1, x_2] \in \mathbb{C}^2$ the subtraction problem corresponds to the function

$$f(x_1, x_2) = x_1 - x_2,$$

and the subtraction algorithm corresponds to the function

$$\tilde{f}(x_1, x_2) = \text{fl}(x_1) \ominus \text{fl}(x_2).$$
We apply the floating point representation axiom, and write

\[ fl(x_1) = x_1(1 + \varepsilon_1), \quad fl(x_2) = x_2(1 + \varepsilon_2) \]

for some \(|\varepsilon_1|, |\varepsilon_2| \leq \varepsilon_{\text{mach}}|.

By the fundamental axiom of floating point arithmetic, we have

\[ fl(x_1) \ominus fl(x_2) = (fl(x_1) - fl(x_2))(1 + \varepsilon_3) \]

for some \(|\varepsilon_3| \leq \varepsilon_{\text{mach}}|.

Combining these results give us

\[
\text{fl}(x_1) \ominus \text{fl}(x_2) = [x_1(1 + \varepsilon_1) - x_2(1 + \varepsilon_2)](1 + \varepsilon_3) = x_1(1 + \varepsilon_1)(1 + \varepsilon_3) - x_2(1 + \varepsilon_2)(1 + \varepsilon_3) = x_1(1 + \varepsilon_4) - x_2(1 + \varepsilon_5),
\]

\[
\tilde{x}_1 - \tilde{x}_2,
\]

for some \(|\varepsilon_4|, |\varepsilon_5| \leq 2\varepsilon_{\text{mach}} + O(\varepsilon_{\text{mach}}^2).

Hence \(\tilde{f}(x_1, x_2) = \tilde{x}_1 - \tilde{x}_2 \equiv f(\tilde{x}_1, \tilde{x}_2),\) where

\[
\frac{|\tilde{x}_1 - x_1|}{|x_1|} = O(\varepsilon_{\text{mach}}), \quad \frac{|\tilde{x}_2 - x_2|}{|x_2|} = O(\varepsilon_{\text{mach}}).
\]

Hence floating point subtraction is a backward stable operation.
We have shown that floating point subtraction is a backward stable operation.

However, from [Lecture#9] we know that subtraction is potentially ill-conditioned:

\[ \kappa(\vec{x}) = \frac{\|J(\vec{x})\|_\infty}{\|f(\vec{x})\|/\|\vec{x}\|_\infty} = \frac{2 \max\{|x_1|, |x_2|\}}{|x_1 - x_2|}. \]

These are NOT contradictory statements!
Example: Inner Product $\vec{x}^* \vec{y}$

Given two vectors $\vec{x}, \vec{y} \in \mathbb{C}^m$, the computed value of the inner product

$$\alpha = \vec{x}^* \vec{y} = \sum_{i=1}^{m} x_i^* y_i$$

is (usually) given by

$$\tilde{\alpha} = (fl(x_1^*) \bowtie fl(y_1)) \oplus (fl(x_2^*) \bowtie fl(y_2)) \oplus \cdots \oplus (fl(x_m^*) \bowtie fl(y_m)).$$

Built from the backward stable fundamental operations in this manner, the inner product is also backward stable. (We leave the proof of this for later).
Example: Outer Product $\vec{x}\vec{y}^*$

Given $\vec{x} \in \mathbb{C}^m$, and $\vec{y} \in \mathbb{C}^n$, the $A \in \mathbb{C}^{m \times n}$ rank-1 outer product is given by

$$A = \vec{x}\vec{y}^* = \begin{bmatrix} x_1\vec{y}^* \\ x_2\vec{y}^* \\ \vdots \\ x_m\vec{y}^* \end{bmatrix}$$

The obvious algorithm is to compute the $mn$ products $x_iy_j^*$ with $\otimes$ and collect the results into the matrix $\tilde{A}$.

This algorithm is **stable, but not backward stable**. — The matrix $\tilde{A}$ will most likely not have rank 1, and can therefore not be written in the form $(\vec{x} + \delta\vec{x})(\vec{y} + \delta\vec{y})^*$.
Example: \((x + C)\)

Let \(C \in \mathbb{C}\) be a fixed non-zero constant, and consider computing \((x + C)\), given \(x \in \mathbb{C}\), we get

\[
\tilde{f}(x) = f1(x) \oplus f1(C) = (x(1 + \varepsilon_1) + C(1 + \varepsilon_2))(1 + \varepsilon_3) = x(1 + \varepsilon_4) + C(1 + \varepsilon_5),
\]

with \(|\varepsilon_1|, |\varepsilon_2|, |\varepsilon_3| \leq \varepsilon_{\text{mach}}, |\varepsilon_4|, |\varepsilon_5| \leq 2\varepsilon_{\text{mach}} + \mathcal{O}(\varepsilon_{\text{mach}}^2)\).

When \(C \neq 0\), and \(x \approx 0\) we are introducing errors of size \(\mathcal{O}(\varepsilon_{\text{mach}})\), independent of \(x\). Relative to the size of \(x\), these errors may become unbounded.

Therefore, we cannot interpret the errors as being caused by small perturbations in the data. Hence \((x + C)\) is not backward stable.
Rule of Thumb:
As a rule, algorithms $\tilde{F} : X \mapsto Y$, where the dimension of $Y$ is greater than the dimension of $X$ are rarely backward stable.

In the outer product example, $X$ has dimension $(m + n)$, and $Y$ has dimension $(m \cdot n)$.

Confusing?
Note that $\tilde{f}(x) = (x + C)$ is not backward stable for fixed $C \neq 0$, but the algorithm for $\tilde{f}(x, y) = (x + y)$ is backward stable.
Example: sin(x) and cos(x)

Floating point calculations of sin(x) and cos(x) are stable, but not backward stable.

Consider sin(x) for \( x = \left( \frac{\pi}{2} - \delta \right) \), \( 0 < \delta \ll 1 \),

\[
\begin{array}{ccccccccc}
\delta & 0.995 & 0.996 & 0.997 & 0.998 & 0.999 & 1 & 1.001 & 1.002 \\
\hline
\delta & 1.5 & 1.55 & 1.6 & 1.65 & 1.7 & 1.75 & 1.8 & 1.85
\end{array}
\]
Suppose we have computed \( \tilde{f}(x) = f_1(\sin(x)) = \sin(x)(1 + \varepsilon_1) \).

Since \( f'(x) = \cos(x) \approx \delta \), we have [Remember Taylor, \( \delta f = f'(x) \delta x \)]

\[
\tilde{f}(x) = f(\tilde{x}) \text{ for some } \tilde{x} \text{ with } (\tilde{x} - x) \approx \frac{1}{\delta}(\tilde{f}(x) - f(x)) = \mathcal{O}\left(\frac{\varepsilon_{\text{mach}}}{\delta}\right).
\]

Since \( \delta \) can be arbitrarily small, the backward error is not of magnitude \( \mathcal{O}(\varepsilon_{\text{mach}}) \).

We have an “exact solution”, but not to a “nearby problem.”
One way of computing the eigenvalues of a square matrix, $A \in \mathbb{R}^{m \times m}$, is through the use of the **characteristic polynomial**

$$p(\lambda) = \det(\lambda I - A).$$

The $m$ roots $\{\lambda_1, \lambda_2, \ldots, \lambda_m\}$, where $p(\lambda_i) = 0$ are the eigenvalues of $A$. Hence, the following algorithm seems reasonable at first glance:

1. Find the coefficients of the characteristic polynomial.
2. Find its roots.

Unfortunately, this algorithm is not only not backward stable, but also unstable; and performs especially badly when the polynomial is expressed in the monomial standard basis $\{x^k\}_{k=0,1,\ldots,m}$.

Even when the eigenvalue problem is well-conditioned, this algorithm may produce answers with large relative errors.
The instability manifests itself in the root-finding step. Recall Wilkinson’s example [Lecture#9], where relative perturbations of the coefficients of

\[ p_{\text{Wilkinson}}(x) = \prod_{i=1}^{20} (x - i) = a_0 + a_1 x + \cdots + a_{19} x^{19} + x^{20} \]

by \( \sim 10^{-10} \) resulted in perturbation of size \( \sim 1-10 \) of the roots.
Example: Eigenvalues of a Matrix

The characteristic polynomial of the diagonal matrix

\[ A_1 = \text{diag}(1, 2, \ldots, 20) \]

is a Wilkinson polynomial or degree 20.

An even simpler example is given by \( A_2 = \text{diag}(1, 1) \), the \((2 \times 2)\)-identity. Trying to find the roots of the characteristic polynomial \( p_2(\lambda) = \lambda^2 - 2\lambda + 1 \), reminds us of the example (also in \[\text{Lecture#9}\]) leading up to Wilkinson’s polynomial:

\[
\begin{align*}
  x^2 - 2x + 1 &= (x - 1)^2 \\
  x^2 - 2x + 0.9999 &= (x - 0.99)(x - 1.01) \\
  x^2 - 2x + 0.999999 &= (x - 0.999)(x - 1.001).
\end{align*}
\]

Where the algorithm above produces errors \( O(\sqrt{\varepsilon_{\text{mach}}}) \).
Example: Eigenvalues of a Matrix

But really... This is a little too pessimistic. IEEE-785-1985 floating point \( \mathbb{F}_{64} \) can represent ("small\( \times \)") integers exactly... But if we try

\[
A = \begin{bmatrix}
1 + 10^{-14} & 0 \\
0 & 1
\end{bmatrix}
\]

with \( p(\lambda) = \lambda^2 - (2 + 10^{-14})\lambda + (1 + 10^{-14}) \), then in an environment where \( \varepsilon_{\text{mach}} = 2.22 \times 10^{-16} \) we get

\[
\{\tilde{\lambda}_1, \tilde{\lambda}_2\} = \{0.99999998509884, 1.00000001490117\}
\]

with errors

\[
\{\tilde{\lambda}_1 - 1, \tilde{\lambda}_2 - (1 + 10^{-14})\} = \{-1.49 \times 10^{-8}, 1.49 \times 10^{-8}\} \sim \mathcal{O}(\sqrt{\varepsilon_{\text{mach}}})
\]

\( \times \) Definition of small in \( \mathbb{F}_{64} \): \( |n| \leq 9, 007, 199, 254, 740, 992 \).
Suppose we have a backward stable algorithm \( \tilde{f} \) for the problem \( f : X \mapsto Y \).

**The Real Question:** Will the results be accurate?

**Answer:** It depends… **on the condition number** \( \kappa = \kappa(x) \).

If \( \kappa(x) \) is small, the results will be accurate. When \( \kappa(x) \) is large, the results may be unreliable.

*We always lose accuracy in proportion to the size of \( \kappa(x) \).*

We make this dependence precise in a theorem…
Theorem (Computational Accuracy)

Suppose a backward stable algorithm \( \tilde{f} \) is applied to solve a problem \( f : X \mapsto Y \) with condition number \( \kappa(x) \) in a floating point environment satisfying the floating point representation axiom, and the fundamental axiom of floating point arithmetic.

Then the relative errors satisfy

\[
\frac{\| \tilde{f}(x) - f(x) \|}{\| f(x) \|} = O(\kappa(x) \epsilon_{mach})
\]

We have tied conditioning, stability, and accuracy together!
Proof (Computational Accuracy)

By the definition of backward stability, we have \( \tilde{f}(x) = f(\tilde{x}) \) for some \( \tilde{x} \in X \), with

\[
\frac{\| \tilde{x} - x \|}{\| x \|} = \mathcal{O}(\varepsilon_{\text{mach}}).
\]

By the definition of \( \kappa(x) \)

\[
\kappa(x) = \sup_{\delta x} \left[ \frac{\| \delta f \|}{\| f(x) \|} \right] \frac{\| \delta x \|}{\| x \|},
\]

we have

\[
\frac{\| \tilde{f}(x) - f(x) \|}{\| f(x) \|} \leq (\kappa(x) + o(1)) \frac{\| \tilde{x} - x \|}{\| x \|} = \mathcal{O}(\kappa(x)\varepsilon_{\text{mach}}). \quad \square
\]

Note: \( o(1) \) is a quantity which converges to zero as \( \varepsilon_{\text{mach}} \to 0 \).
Backward Error Analysis

The method of proof we used defines the strategy for *backward error analysis*.

We obtain the accuracy estimate in two steps:

1. Analyze the **condition** of the problem.
2. Analyze the **stability** of the algorithm.

**Conclusion:** If the algorithm is backward stable, then the accuracy is proportional to the condition number.

At this point, this may seem natural and straight-forward?

Naively, *Forward Error Analysis* may seem like a tempting alternative...
At first glance, the most natural form of error analysis is to apply the *floating point representation axiom*, and the *fundamental axiom of floating point arithmetic* directly to the algorithms and

1. Introduce error bounds on each operations.
2. Track how the errors compound throughout the computation.

It turns out that this approach is very difficult to carry out successfully.

Here there is no separation of algorithm and problem; hence the forward error analysis must capture both the stability behavior of the algorithm, and the conditioning of the problem. How do we “detect” the conditioning in operation-level analysis?!?
Backward Error Analysis is the right tool: in general, the best algorithms for a problem will compute the exact solution to a slightly perturbed problem. The method of backward error analysis is perfectly tailored to this slightly “backward view.”
We carefully analyze the stability of two of our most important algorithms:

- The Householder Triangularization algorithm for computing the QR-factorization.
- The back (and forward) substitution algorithm.

Together they are the foundation upon with we build our solvers for $A\vec{x} = \vec{b}$ for both square and non-square $A$.

Then, we re-visit the Least Squares problem — and carefully look at the conditioning of the problem, and stability of the algorithms we use for solving the problem.