Math 128A, Summer 2019

Lecture 3, Wednesday 6/26/2019

CLASS ANNOUNCEMENTS:
Instructions/Tips for problem sets,

- different page for each problem
- don't try to save paper, (write large)
- don't type (unless you check it a lot)
- include code and output plots
- credit your sources
- don't use proof by contradiction, because it's easy in numerical analysis and **constructive mathematics**, it's easy to make mistakes and hard to find them; additionally, proofs by contradiction *rarely illuminate* the underlying ideas
- be self-critical (review your work; the best growth in Math is when you find your own mistakes)

Today's topics:

- Floating Point
- Algorithms
- big-O notation

1 Floating Point Arithmetic

Recall the Mantra: "Binary Floating Point operations $\{+, -, \times, /, \sqrt{\ }\}$ on Floating Point numbers deliver **the exact result, correctly rounded**."

Quantitatively, this is equivalent to (or implies) that

$$\begin{aligned}
fl(x+y) &= (x+y)(1+\delta), \qquad |\delta| \leq \frac{\varepsilon}{2} \\
fl(x-y) &= (x-y)(1+\delta), \qquad |\delta| \leq \frac{\varepsilon}{2} \\
fl(x\times y) &= (x\times y)(1+\delta), \qquad |\delta| \leq \frac{\varepsilon}{2} \\
fl(x/y) &= (x/y)(1+\delta), \qquad |\delta| \leq \frac{\varepsilon}{2} \\
fl(\sqrt{x}) &= (\sqrt{x})(1+\delta), \qquad |\delta| \leq \frac{\varepsilon}{2}
\end{aligned}$$

Remark: Floating point arithmetic is **NOT** associative.

That is,

$$fl[\underbrace{(x+y)}_{\text{this error gets compounded}} + z] \neq fl[x + (y+z)]$$

Rule 1: Don't compute 0.

Rule 2: Minimize (the values of) intermediate results if possible.

2 Algorithms

Example: Compute the sum of n numbers: S_n.

```
function [S] = sumn(X,n)

% X is array of values
% n is how many terms we want to add

S = 0;
for j = 1:n
  S = S + X(j);
end

end
```

How much did this algorithm cost? Cost of `sumn(X,n)` is n adds, or "flops".

Definition: Big-O -

We say $f(n) = O(g(n))$ to be equivalent to $\exists K, N$ with

$$|f(n)| \leq Kg(n)$$

for $n \geq N$.

Example:

$$\begin{aligned} 17n^2 &= O(n^2 \log n) \\ n^2 &= O(2^n) \\ 10^{17}n^3 &= O(1.0000001^n) \end{aligned}$$

Aside:

$$\begin{aligned} O(n) &= O(n \log n) \\ &= O(n|\log \varepsilon|) \end{aligned}$$

The floating point result for calulating

$$\begin{aligned} fl(S_n) &= fl[\underbrace{fl(S_{n-1})} + X_n] \\ &\quad \text{floating point result of this algorithm is well defined} \\ &= [fl(S_{n-1}) + X_n](1+\delta_n), \qquad |\delta_n| \leq \frac{\varepsilon}{2} \\ fl(S_n) - S_n &= fl[S_{n-1}](1+\delta_n) - S_n + x_n(1+\delta_n) \\ &= fl(S_{n-1})(1+\delta_n) - S_{n-1}(1+\delta_n) + S_{n-1}\delta_n + X_n\delta_n \\ &= fl[S_{n-1}](1+\delta_n) - S_{n-1}(1+\delta_n) + S_n\delta_n \\ &= [fl(S_{n-1}) - S_{n-1}](1+\delta_n) + \delta_n S_n \\ E_n &= E_{n-1}(1+\delta_n) + \delta_n S_n \end{aligned}$$

where δ_n is a nice error, relative to what we are trying to compute.

Then,

$$
\begin{aligned}
|E_n| &\le |E_{n-1}|\left(1+\frac{\varepsilon}{2}\right)+\frac{\varepsilon}{2}|S_n| \\
|E_{n-1}| &\le |E_{n-2}|(1+\frac{\varepsilon}{2})+\frac{\varepsilon}{2}|S_{n-1}| \\
|E_n| &\le |E_{n-2}|\left(1+\frac{\varepsilon}{2}\right)^2+\frac{\varepsilon}{2}\left(1+\frac{\varepsilon}{2}\right)|S_{n-1}|+\frac{\varepsilon}{2}|S_n| \\
&\le \ddots \\
&\le |E_1|\underbrace{\left(1+\frac{\varepsilon}{2}\right)^{n-1}}+\frac{\varepsilon}{2}\underbrace{\left(1+\frac{\varepsilon}{2}\right)^{n-2}}|S_2|+\frac{\varepsilon}{2}\underbrace{\left(1+\frac{\varepsilon}{2}\right)^{n-3}}|S_3|+\cdots+\frac{\varepsilon}{2}|S_n| \\
&\text{these act like } O(n\varepsilon^2) \\
&\le \frac{\varepsilon}{2}\left(|S_1|+|S_2|+\cdots+|S_n|\right)+O(n\varepsilon^2)
\end{aligned}
$$

We assert (as it's true) that we compute S_2, then S_3, etc, and calculate and store along the way.

Algorithm for e^x
Let $e^x := e^{m+r} = [e^m][e^r]$, where $m \in \mathbb{Z}$ and $r = x - m$.

Range reduction:

1. store e
2. compute e^m; take m, turn it into binary and

$$m = \sum_{k=0}^{N} b_k 2^k$$

and

$$e^{2^k} = \left(e^{2^{k-1}}\right)^2$$

is in $\log m$ time.

```
function y = expx(x)
% x is the exponent of e
% we won't take a tolerance for precision for now

  m = round(x) % nearest integer to x
```

$$|r| \le 1/2, \quad k! \approx \left(\tfrac{k}{e}\right)^k \implies \tfrac{r^k}{k!} \approx \left(\tfrac{e}{2k}\right)^k = 10^{-15}$$

$$k \ge 15$$

should be enough

$$
\begin{aligned}
|r| = |x-m| &\le 1/2 \\
e^r &= \sum_{k=0}^{q}\frac{r^k}{k!}+O\left(\frac{e}{2q}\right)^q \\
e^r &= 1+\frac{1}{1!}+\frac{r^2}{2!}+\frac{r^3}{3!}+\cdots
\end{aligned}
$$

Remark: When summing a Taylor Series, we typically sum from right to left, for smaller values first, storing, then adding bigger terms as we go.

Definition: Horner's Rule -

$$e^r = 1 + r(\frac{1}{1!} + r(\frac{1}{2!} + r(\frac{1}{3!} + \cdots)))$$

We compute in $\rightarrow$ out.

```
oursum = 1 / (q!)
for i = (q-1):(-1):0
  oursum = 1 / (i!) + r * oursum
end

y = oursum * (e^m)
```

Example: Computing this is difficult:

$$\frac{e^x - 1}{x}$$

as the numerator nears 0 for small x, and so does the denominator

3 Convergence, Characterizing Error

Definition: Rate of Convergence -

An algorithm producing $p_n \approx p$. If

$$\left|\frac{p_n - p}{p}\right| \leq O\left(n^{-r}]\right)$$

The rate of convergence is $O(n^{-r})$, for example "second-order" $r = 2$, $O(n^2)$.

E.g.

$$e^r - \sum_{k=0}^{q} \frac{r^k}{k!} = O\left[\left(\frac{e}{2q}\right)^q\right]$$

, where big-O is the rate of convergence.

Example: Approximating a derivative. Define:

$$D_h f(x) = \frac{f(x+h) - f(x)}{h} = O(h^{\text{what order?}})$$

(1) one way is MVT:

$$f(x+h) - f(x) = f'(c)h, \qquad \text{for some } x \leq c \leq x+h$$

MVT is a bit too crude here, so we probably want to use one more term in the Taylor expansion.

(1, modified)

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(c)h^2$$

Then we have

$$D_h f(x) := f'(x) + \frac{1}{2}f''(c)h$$

So the Rate of Convergence (RoC) is $O(h^1) = O(1)$, and we call the "Order of convergence" 1.

Remark: Ok, great, now we have some idea of algorithms and cost. What happens in floating-point arithmetic? In the above section, we assume that our approximations are exact.

4 Rate of Convergence in Floating Point Arithmetic

Example:

$$\begin{aligned}
fl[f(x)] &= f(x)(1+\delta_1) \\
fl[f(x+h)] &= f(x+h)(1+\delta_2) \\
fl[F_h f(x)] &= \frac{[f(x+h)(1+\delta_2) - f(x)(1+\delta_1)]\,(1+\delta_3)}{h}(1+\delta_4) \\
&= \frac{f(x+h) - f(x)}{h} + \frac{f(x+h)(3\delta_5)}{h} + \frac{f(x)(3\delta_6)}{h} \\
&= f'(x) + O(h) + O\left(\frac{\varepsilon}{h}\right)
\end{aligned}$$

Don't let $h \to 0$, due to the last term. How do we deal with this? There's a sweet spot in the middle somewhere, where $O(h)$ doesn't explode and $O(\frac{\varepsilon}{h})$ doesn't explode. To find this, we suppose:

$$O(h) + O(\frac{\varepsilon}{h}) := O(1) + O(\frac{-\varepsilon}{h^2})$$

where $h^2 \approx \varepsilon \implies h \approx \sqrt{\varepsilon}$.

So the best achievable error is about:

$$= O(\sqrt{\varepsilon}) + O\left(\frac{\varepsilon}{\sqrt{\varepsilon}}\right) = O(\sqrt{\varepsilon}) = O(10^{-7})$$

Centered 2nd-order approximation: If

$$D_h^2 f(x) = \frac{f(x+h) - f(x-h)}{2h} = f'(x) + O(h^2)$$

Then $\min O(h^2) + O(\frac{\varepsilon}{h}) \implies h = \frac{\varepsilon}{h^2} \implies h^3 = \varepsilon \implies h = \varepsilon^{1/3}$.

So the best achievable error is at

$$O(\varepsilon^{2/3}) + O(\frac{\varepsilon}{\varepsilon^{1/3}}) = O(\varepsilon^{2/3}) = O(10^{-10})$$

Remark: Today's lecture ends today. Tomorrow we'll talk about bisection.

5 Notes from Textbook

Definition: linear, exponential (growth of error) -

- If $E_n \approx CnE_0$, where C is a constant independent of n, then the growth of error is said to be **linear** and **stable**.
- If $E_n \approx C^n E_0$, where $C > 1$, then the growth of error is called **exponential** and **unstable**.

Remark: Linear growth of error is usually unavoidable, and results are generally acceptable when C and E_0 are small.

Definition: Rate (Order) of Convergence -

Suppose $\{\beta_n\}_{n=1}^{\infty}$ is a sequence known to converge to zero and $\{\alpha\}_{n=1}^{\infty}$ converges to a number α. If a positive constant K exists with

$$|\alpha_n - \alpha| \leq K|\beta_n|, \quad \text{for large } n,$$

then we say that $\{\alpha_n\}_{n=1}^{\infty}$ converges to α with **rate/order of convergence** $O(\beta_n)$. We read this "Big Oh of Beta N". In text, we write it as $\alpha_n = \alpha + O(\beta_n)$.

Remark: Although our definition permits $\{\alpha_n\}_{n=1}^{\infty}$ to be compared with an ARBITRARY sequence $\{\beta_n\}_{n=1}^{\infty}$, in nearly every case we use

$$\beta_n = \frac{1}{n^p},$$

for some number $p > 0$. We are generally interested in the LARGEST value of p with $\alpha_n = \alpha + O(\frac{1}{n^p})$.