Contents

1 Lecture 1

1.1 Significant Digits:

Suppose a problem has a true answer $y = 0.0012345...$, and an algorithm returns $\hat{y} = 0.001227...$, then we have only *two significant digits* of accuracy.

The **Relative Error** is given by

$$\varepsilon = \frac{|\hat{y} - y|}{|y|}$$

and the number of significant digits is then given by

$$\log_{10} \frac{1}{\varepsilon} = \left| \log_{10} \varepsilon \right|$$

However, usually we don't know y. How do we assess ε for $\hat{y}$?

- Sometimes $\exists$ a specific test, e.g. solve $Ax = b$ and check $||Ax - b||_2 \to \varepsilon$.
- If the algorithm has an *effort controller parameter*, e.g. n, we can create a sequence $\hat{y}_n$, for $n =$ small,...,larger, and look for the convergence of this sequence.

Example: Suppose $y = \frac{1}{1^2} + \frac{1}{2^2} + ... = \sum_{k=1}^{\infty} k^{-2}$, and we have that $\hat{y}_n = S_n = \sum_{k=1}^{n} k^{-2}$. If we run basic code to calculate the sums for increasing values of n, we can see that $\varepsilon \approx \frac{1}{n}$. This is called *algebraic convergence* or convergence of *order 1.*

We can determine significant digits of the output from our code by examining the *freezing of digits.* We will assume that as n gets larger, our solution approaches the exact solution, and thus increasing n returns a more accurate solution. We can then say that if the first k digits of the output of n are the same as the first k digits of the output of $n + 1$, then we have k significant digits for the output of n.

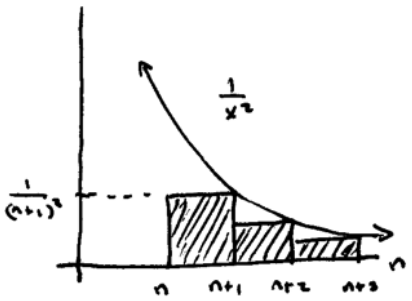

Figure 1.1: N-Term Error

The N-term error is given by $\varepsilon_N = \sum_{k>N} \frac{1}{k^2}$. We can bound this error by considering its Riemann sum and bounding it by a function as shown in Figure 1.1. If we take the integral of this function, we have that $\varepsilon \leq \int_n^{\infty} \frac{1}{x^2} dx = \frac{1}{n}$. Thus we have shown that $\varepsilon \leq \frac{1}{n}$.

1.2 O-Notation (Asymptotic Behavior)

Big O-notation and little o-notation describe the limiting behavior of functions, characterizing them by their growth rates.

1.2.1 Big O-notation

The previous example had asymptotic behavior of $\varepsilon = O\left(\frac{1}{n}\right)$ as $n \to \infty$. In general, we say that $f(n) = O\Big(g(n)\Big)$ if $\exists\, C, N$ s.t. $\frac{f(n)}{g(n)} \leq C \;\forall\; n \geq N$. In big O-notation, we can get rid of the junk at the beginning of a function, as we are only concerned with the value as $n \to \infty$ (Figure 5.1).

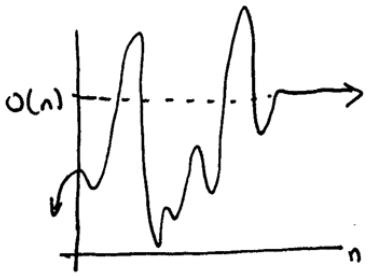

Figure 1.2:

Example: Is $n^2 \ln n = O(n^2)$ as $n \to \infty$?

If we consider the ratio: $\frac{n^2 \ln n}{n^2} = \ln n \leq C$, because $\ln n$ is unbounded, $\nexists\, C, N$ s.t. this holds $\forall\; n \geq N$.

Example: What is the smallest O for $\frac{n}{n^2-1}$?

We will simplify this problem by replacing our expression with another similar one. However, we cannot consider $\frac{n}{n^2}$ because $\frac{n}{n^2} < \frac{n}{n^2-1}$, thus giving us a lower bound but not an upper bound. Since we need an upper bound, two other options are as follows:

- Because $n^2 - 1 \geq n^2 - n$, we have that $\frac{n}{n^2-1} \leq \frac{n}{n^2-n} = \frac{1}{n-1}$, thus giving us $\frac{n}{n^2-1} = O(\frac{1}{n-1})$. However, we can do better than that.
- We can also see that $n^2 - 1 \geq \frac{n^2}{2} \;\forall\; n \geq 2$, and thus $\frac{n}{n^2-1} \leq \frac{n}{n^2/2} = \frac{2}{n}$. If we let $C = 2$, and we then have $\frac{n}{n^2-1} = O(\frac{1}{n})$.

1.2.2 Error vs. Effort (complexity):

There are two ways in which we will consider big O-notation, for a measure of error, and for a measure of algorithmic complexity or necessary effort by the algorithm. Figure 3.2 shows an example plot for each of these purposes.

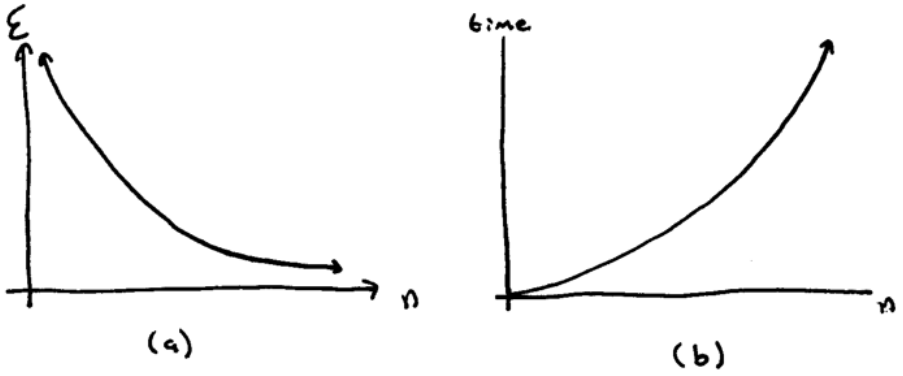

Figure 1.3: (a) $\varepsilon = O(\frac{1}{n})$ for error ε. (b) $T = O(n^3)$ for effort or complexity over time T.

Example: Given error $= \varepsilon = O(\frac{1}{n})$ and effort $= T = O(n^3)$, we can then compare ε and T.

1.2.3 Little o-notation

Little o-notation is a stronger statement that big O-notation. We say that $f(n) = o\Big(g(n)\Big)$ if $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = 0$.

Example: Is $e^{-n} = o(\frac{1}{n})$?

If we consider the limit definition, we have

$$\lim_{n\to\infty} \left|\frac{e^{-n}}{n^{-1}}\right| = \lim_{n\to\infty} \frac{n}{e^n}.$$

Using L'Hopital's rule gives us

$$\begin{aligned}\lim_{n\to\infty} \frac{n}{e^n} &= \lim_{n\to\infty} \frac{1}{e^n} \\ &= 0.\end{aligned}$$

Note we can do the same thing asking if $e^{-n} = o(n^{-2})$. Using L'Hopital's rule twice gives us

$$\lim_{n\to\infty} \left|\frac{e^{-n}}{n^{-k}}\right| = \lim_{n\to\infty} \frac{n^k}{e^n} = ... = \lim_{n\to\infty} \frac{k!}{e^n} = 0.$$

Using induction to show that $e^{-n} = o(n^{-k})$ for any k gives us the following:

Lemma: e^{-n}, or r^n for any $|r| < 1$, vanishes faster than **any** algebraic order. If $\varepsilon = O(e^{-cn})$ for $c > 0$ or $\varepsilon = O(r^n)$ for $|r| < 1$, we call it *exponential convergence* or *linear convergence*.

2 Lecture 2

2.1 Complex Basics

If we consider a point in the complex plane, e.g. $z = 2 + 3i$. We then have that the *magnitude* $r = |2 + 3i| = \sqrt{2^2 + 3^2}$ and the *phase* $\theta = \arctan(\frac{3}{2})$. Note, θ can be found in Matlab using the command $angle(2 + 3i)$. It is also helpful to use $1i$ for an imaginary number in Matlab, because i is often a variable.

2.1.1 Identities:

- $e^{i\theta} = \cos(\theta) + i\sin(\theta)$
- $\log n < n^k < e^n \ \forall\ k > 0$ as $n \to \infty$
- $a^5 + a^6 + ... = a^5(1 + a + ...)$

2.2 Convergence

2.2.1 Algebraic Convergence

2.2.2 Exponential/Linear Convergence

This is the name of convergence for an error

$$\varepsilon_n := \frac{|\hat{y} - y|}{|y|} = O(r^n)$$

for some *convergence rate* $r \in \{0, 1\}$.

Examples:

- Consider a Taylor Series centered at $x = 0$,

$$-\ln(1-x) = x + \frac{x^2}{2} + \frac{x^3}{3} + ...$$

Let us consider the tail of the series, where the n^{th}-term error is given by $\varepsilon_n = \sum_{k>n} \frac{1}{k} x^k$. We can compare this to the geometric series of

$$\begin{aligned} \sum_{k>n} x^k &= x^{n+1} \sum_{k=0}^{\infty} |x|^k \\ &\geq \varepsilon_n. \end{aligned}$$

We also know that $\sum_{k=0}^{\infty} |x|^k = \frac{1}{1-|x|}$. Thus we have that $\varepsilon = O(|x|^{n+1}) = O(|x|^n)$ because the constant gets absorbed into the C. It is important to note that there is exponential convergence only for $|x| < 1$. In this case, $r = |x|$.

Suppose we need $n = 100$ to get $\varepsilon = 10^{-8}$, what n will give us $\varepsilon = 10^{-16}$? If $r^n \approx 10^{-8}$, then $r^{2n} = 10^{-8*2} = 10^{-16}$. Thus we need $n = 200$.

- Consider $f(x) = \frac{1}{1+x^2}$. Note that f is *smooth*, which means that $f \in C^\infty$, or f is continuous and infinitely complex differentiable. To write this function as a geometric series, let us rewrite f as

$$f(x) = \frac{1}{1+x^2} = \frac{1}{1-(-x^2)} = \sum_{i=1}^{k} (-x^2)^k$$

Just as before, we can see that this series converges for $|x| < 1$, and diverges otherwise.

We know that the size of the tail of this function must be less than the sum, i.e. for some N, $|tail| < \sum_{k>N} |x^{2k}| = O(|x|^{2n})$.

However, this $O(|x|^{2n})$ is misleading, as we are dealing with a special taylor series without odd points, i.e. $1 - x^2 + x^4 - ....$ If we include odd terms for a generic, non-zero x_0, then $r = |x|$, the same as we had in the first example for $-\ln(1-x)$. Convergence is as if $\exists$ a singularity of distance 1 from our expansion center, $x_0 = 0$, i.e. $f \to \infty$ as $(1 + x^2) \to 0$. Thus $x^2 = -1$, and this singularity is at $x = \pm i$.

Theorem: Asymptotically, the rate of convergence in a series is given by the ratio of the distance from the center to the evaluation point to the distance from the center to the nearest singularity. I.e. suppose we center our series at a, evaluate it at x, and d_s is the distance to the closest singularity, then for large n

$$r = \left| \frac{x-a}{d_s - a} \right|$$

Corollary: A Taylor series converges exponentially fast for points x closer to the expansion point x_0, than the distance from x_0 to the closest singularity in $\mathbb{C}$, i.e. $|r| < 1$.

Theorem: Let f be analytic in some disc $|x - x_0| \leq R$, where the distance to the nearest singularity is $> R$. Then the error in the n-term Taylor series at x is $\varepsilon = O\left(\left|\frac{x-x_0}{R}\right|^n\right)$.

2.2.3 Super Exponential Convergence:

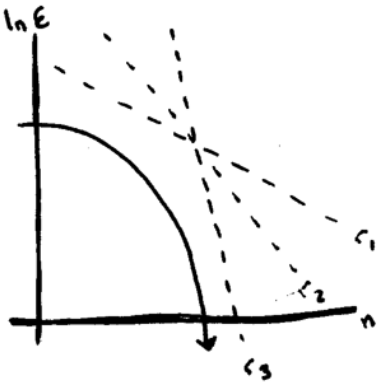

Figure 2.1: N-Term Error

Consider the Taylor series for e^x about $x = 0$,

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots,$$

with error rate $\varepsilon = \sum_{k>n} \frac{x^k}{k!}$. Here we prove that this series is *super-exponentially* convergent, i.e. for each rate $r > 0$, no matter how small, ε, the error in truncating to n terms, is $O(r^n)$. This implies that on a semi-log-y plot ($\log \varepsilon$ vs n), the graph has increasingly negative slope, that can become arbitrarily steep.

Proof. Given location x, and rate $r > 0$, then choose integer $N > |x|/r$. Then, for $k \geq N$, a single term has the bound

$$\frac{|x^k|}{k!} \leq \frac{N^N}{N!} \cdot \frac{|x|^k}{N^k} \leq C r^k$$

(Why? Make sure you understand this. C can be large but is always constant with respect to k.) So the tail is bounded as usual by $\varepsilon \leq C \sum_{k>n} r^k = O(r^n)$ for all $n \geq N$. □

Note that this arbitrarily high rate of exponential convergence corresponds to e^x being analytic in arbitrarily large discs around the origin. This is called an *entire* function.

3 Lecture 3

3.1 Newton Iterations

Example: Computing $x = \sqrt{y}$, for $y \geq 0$. Guess some $x_0 > 0$, and iterate

$$x_{n+1} = \frac{x_n + \frac{y}{x_n}}{2}$$

for $n = 0, 1, \ldots$. If we test, we can see that this algorithm doubles the number of correct digits each iteration, which we call *quadratically convergent*. This algorithm is actually a dynamical system, which converges to a *fixed point* at $\sqrt{x}$, and also a case of a Newton Iteration.

Def: More rigorously, Newton's iterations are a numerical technique to find roots of a function, f, through iterations

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

and quadratic convergence is error s.t.

$$\frac{\varepsilon_{n+1}}{e_n^2} \leq C \text{ for some } C$$

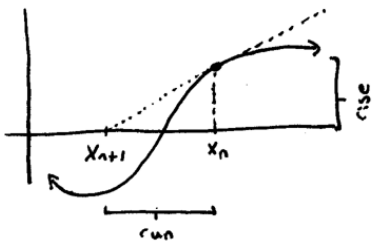

Figure 3.1: N-Term Error

Taylor's Theorem: Given an expansion point a and evaluation point x for some function f, we have

$$f(x) = f(a) + f'(a)(x-a) + ... + \frac{f^{(n)}(a)}{n!}(x-a)^n + R$$

where the $R = \frac{f^{(n+1)}(\hat{a})}{(n+1)!}(x-a)^{n+1}$ is the remainder term, and $\hat{a} \in (a, x)$.

Theorem: Let $f \in C^2$ and $f'(z) \neq 0$. If x_0 is sufficiently close to z, then

$$\lim_{n\to\infty} \frac{|x_{n+1} - z|}{(x_n - z)^2} = C$$

This is equivalent to saying f is quadratically convergent. This does not just give a bound on the error, it gives a rate of decay. Note, this is a root finding theorem, not a continuation of Taylor's Theorem.

- For each $C' > C$, $\exists$ an interval I symmetric about z where $\left|\frac{f''(z)}{2f'(z)}\right| < C'$. If $x_n \in I$ then $|x_n - z| < \frac{1}{C'}$, and from Taylor's Theorem we have the inequality

$$|x_{n+1} - z| < C'(x_n - z)^2 < C'\frac{1}{C'}|x_n - z|$$
$$|x_{n+1} - z| < C'(x_n - z)^2 < |x_n - z|$$

  This is equivalent to saying that the iteration gets closer to the root each time if $x_n \in I$.

3.2 Floating Point Representation

This is a fancy way of saying the computer only works to finite precision, e.g. compute $(1 + 10^{-17}) - 1 = ?$ Mathematically this should give us 10^{-17}, but it will likely give us zero. However, suppose we try $(10^{-6} + 10^{-17}) - 10^{-6} = ?$ This will actually give us (approximately) the correct answer, 10^{-17}. This is due to *rounding error*, where (Matlab) will store only 16 digits of relative accuracy, and thus relative changes of less than 10^{-16} are not recorded.

Example: Consider $\sum_{k=1}^{\infty} \frac{1}{k^2} = \zeta(2)$. Starting at $k = 10^8$, each additional term will have no effect on the computational sum because if $k \geq 10^8$, $\frac{1}{k^2} \leq 10^{-16}$. Thus all terms for $10^8 \leq k < \infty$ are not included in a computational sum. To solve this problem, we will *reverse the order of the sums*. If we start our sum at zero, values of $\frac{1}{k^2} \leq 10^{-16}$ are still computationally relevant. Thus we will sum from the smallest values to the largest. Note, we must choose some number N to be the upper bound, $\sum_{k=N}^{1} \frac{1}{k^2}$.

3.2.1 IEEE Double Precision

Let $x \in \mathbb{R}$ be represented by some finite number of bits. x is a finite subset of $\mathbb{R}$, and must have a highest and lowest possible value, the standard of which is $\pm 10^{308}$. The standard method of representing numbers in computing is IEEE Double Precision. For a base β and precision t, we have a *set of floating point numbers*:

$$F = \{\pm \frac{m}{\beta^t}\beta^e, 0, \pm\infty, \text{NAN}\}$$

The standard base is $\beta = 2$ and $t = 53$, where $e \in \{-1022, -1021, ..., 1023\}$. The *mantissa* m represents the precision bits of the number

- The last two numbers in the set $\pm\infty, \text{NAN}$ are considered special types of numbers, and do not behave as other numbers.
- Given 64 bits, we use 52 bits for the mantissa m, 11 bits for the exponent e, and one bit for the sign. Note, 64 bits = 8 bytes.
- *double precision* $[1, 2]$ is represented by $\{1, 1 + 2^{-52}, 1 + 2 \cdot 2^{-52}, ..., 1 + 2^{52} \cdot 2^{-52} = 2\}$. There are gaps in here! The biggest relative gap is the first one, where the boundaries are smallest, with a relative gap between $(1, 1+2^{-52})$ given by $2.2 \cdot 10^{-16}$. (Note, reference 16 bit).
- The interval $[2, 4]$ is represented by twice the size of intervals as used for $[1, 2]$. The interval $[-2, -1]$ is represented by negative versions.
- The relative error is the same, at the beginning of each interval.

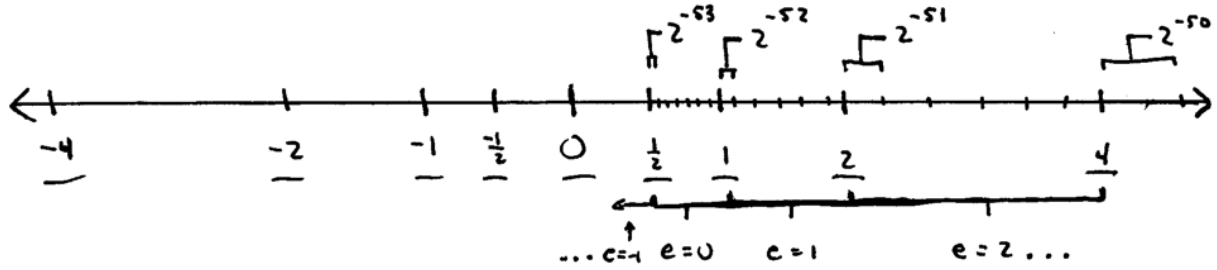

Figure 3.2:

eps $= 2.2 \cdot 10^{-16}$.

$\varepsilon_{mach} = \frac{eps}{2} = \frac{1}{2}\beta^{1-t}$ = largest allowed rounding error. I.e. any number you give a computer will be rounded to the closest number in its set, and ε_{mach} is the maximum relative rounding error. $\forall\ x \in \mathbb{R}, \exists\ x' \in Fs.t.\frac{x'-x}{|x|} \leq \varepsilon_{mach}$.

ε_{mach} is machine precision. GAP after 1 is $eps = 2\varepsilon_{mach}$.

4 Lecture 4

4.1 Rounding Error

A *flop* is a floating point operation. We have the following important rules of floating point, where $fl(x)$ is the floating point representation of x:

- any input is rounded to the nearest member of F.
  - $fl(x) = (1+\varepsilon)x$ for some $|\varepsilon| \leq \varepsilon_{mach}$.
  - Worst case scenario is error of $eps/2$. This is why machine precision $\varepsilon_{mach} = eps/2 = 1.1 \cdot 10^{-16}$.
- Each arithmetic operation ('flop') done by the machine is done at worst ε_{mach} relative error. I.e. let $x, y \in F$, then $x ++ y = (1+\varepsilon)(x+y)$ for some $|\varepsilon| \leq \varepsilon_{mach}$, where $++$ is done by the machine. Same for $\times, /$, and some built in functions.

  Often the cumulative error is small, but not always. A good algorithm should have it as small as reasonable.

Example: Subtraction. Let $x = 1 + 10^{-10}, y = 1$. Then

$$\begin{aligned} fl(x) &= (1+\varepsilon_1)(1+10^{-10}) \\ fl(y) &= (1+\varepsilon_2)1 \end{aligned}$$

Note, we know that $\varepsilon_2 = 0$, but this is not always the case. Then

$$\begin{aligned} fl(x) - fl(y) &= (1+\varepsilon_1)x - (1+\varepsilon_2)1 \\ &= x - 1 + \varepsilon_1 x - \varepsilon_2 \end{aligned}$$

The answer we are looking for is given by $x - 1$, and we have cumulative error $\varepsilon_1 x - \varepsilon_2$, where $|\varepsilon_1|, |\varepsilon_2| \leq \varepsilon_{mach}$. The worst case scenario is $\varepsilon_{tot} = (x+1)\varepsilon_{mach}$ if $\varepsilon_1 = \varepsilon_{mach}$ and $\varepsilon_2 = -\varepsilon_{mach}$. This error is approximately equal to $2\varepsilon_{mach} = eps$, with relative error given by

$$\begin{aligned} \frac{2\varepsilon_{mach}}{10^{-10}} &= 2 \cdot 10^{10} \cdot \varepsilon_{mach} \\ &\approx 2 \cdot 10^{-6} \end{aligned}$$

However, we have still only dealt with initial rounding errors. Now we must consider flop error, where $(fl(x) - fl(y))(1-\varepsilon_3)$. $(1+\varepsilon_3)$ gives a maximum relative error of 10^{-16}, which is negligible because it will be dominated by the potential rounding error of 10^{-6}. This error is called *catastrophic cancellation*

Example: Evaluate $y = \frac{1-\cos(x)}{x^2}$ at $x \approx 0$ (preferably to high relative precision). The answer to $\cos(x)$ is only accurate to ε_{mach} relative error. Thus the numerator can be given by $1 - \cos(x) + \varepsilon_1$, $|\varepsilon_1 \leq |\varepsilon_{mach}$. Introducing rounding error and calculation error in the denominator, gives us

$$\begin{aligned} \hat{y} &= \frac{1 - \cos(x) + \varepsilon_1}{\Big(x(1+\varepsilon_2)\Big)^2 (1+\varepsilon_3)} \\ &= \frac{1 - \cos(x) + \varepsilon_1}{x^2}(1+3\varepsilon) \end{aligned}$$

where 3ε is a sum of $\varepsilon_2, \varepsilon_3$. Let us note that $\frac{1}{1+\varepsilon_1} \approx 1 + \varepsilon_1$. We then have that

$$\text{abs error} = \frac{\varepsilon_1}{x^2} + .5(3\varepsilon)$$

$.5(3\varepsilon)$ is very small. But $\frac{\varepsilon_1}{x^2}$ can become very bad, e.g. $x = 10^{-6}$ gives an absolute error of $\frac{\varepsilon_{mach}}{(10^{-6})^2} \approx 10^{-4}$, especially if the sign of the answer is $\approx .5$.

How can we fix this? Do not evaluate the cosine and subtract 1. Instead, note that $1 - \cos(x) = 2\sin^2(\frac{x}{2})$, and thus $y = \frac{2\sin^2(\frac{x}{2})}{x^2}$. Thus produces a much more accurate result, one of relative error of machine precision, $O(\varepsilon_{mach})$, because $\sin(x)$ is computed to high relative error. When we evaluated $\cos(x)$, we subtracted the majority of it leading to a high error.

Example: Let us consider finding the roots of $ax^2 + bx + c = 0$. if $ac << b^2$, the discriminant is dominated by b^2, and will look like $b+$ something very small relative to b. There are two roots,

$$x_+ = \frac{1}{2a}(-b + (b + small))$$
$$x_- = \frac{1}{2a}(-b - b - small)$$

For x_+, the large part of the root b is subtracted off, leaving a small level of accuracy in the small part. For x_-, this is not a problem, because the small part is still arbitrarily small with respect to the root.

How can we fix this? Rewrite our quadratic formula as

$$x^2 + \frac{b}{a}x + \frac{c}{a} = (x + x_-)(x + x_+) = 0$$

thus telling us that we can compute x_- accurately, and then compute $x_+ = \frac{c}{ax_-}$.

4.2 Conditioning of a problem

Is it reasonable to demand $\sin(10^{16})$ to high accuracy? If $x = 10^{16}$, then $fl(x) = (1+\varepsilon)x$, where $|\varepsilon| \leq 1$. This is HUGE error in rounding! Evaluating $\sin(fl(x))$ could thus be very inaccurate. This problem is called *ill-conditioned*.

Given a math problem, e.g. evaluate $y = f(x)$, how sensitive is the output to changes in input? We say that f is *well-conditioned* if small changes δx in the input x cause 'small' changes in output, $\delta f := f(x + \delta x) - f(x)$. The *absolute condition number* is given by

$$\tilde{\kappa}(x) = \frac{|\delta f|}{|\delta x|}, \qquad \delta x << 1$$

The *relative condition number* is essentially the ratio of relative jiggle on input to relative jiggle on output. Under the assumption that $\frac{\delta f}{\delta x} \approx f'(x)$, the relative condition number is given by

$$\kappa(x) = \frac{|\delta f|/|f|}{|\delta x|/|x|}, \qquad \delta x << 1$$
$$= \left|\frac{xf'(x)}{f(x)}\right|$$

A small value of κ implies that the problem is well-conditioned, whereas a large κ implies an ill-conditioned problem.

Examples:

- Is $f(x) = 2x$, $f'(x) = 2$ well conditioned?
  $\kappa = \left|\frac{2x}{2x}\right| = 1$. It is well conditioned for all x.
- Is $f(x) = e^x$, $f'(x) = e^x$ well conditioned?
  $\kappa = \left|\frac{xe^x}{e^x}\right| = |x|$. This is well-conditioned for small x.
- When is $f(x) = \sin(x)$ ill-conditioned?
  $\kappa = \left|\frac{x\cos(x)}{\sin(x)}\right| = \left|\frac{x}{\tan(x)}\right|$. This is ill-conditioned at $x = n\pi \ \forall \ n$ and for most large x, except when $\tan(x) \approx \infty$.

5 Lecture 5

5.1 Finite-Differencing to Evaluate Derivatives

Goal is to approximate $f'(x)$ at a certain point if all you have access to is the evaluations of $f(x)$.

Example: $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$ for small h. What is the error vs. h? Use Taylor's Theorem about x.

$$\begin{aligned} d &= \frac{1}{2h}\Big(f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{3!}f'''(q)\Big)... \\ &\qquad - \frac{1}{2h}\Big(f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{3!}f'''(q')\Big) \\ &= \frac{1}{2h}\Big(2hf'(x) + \frac{h^3}{3!}(f'''(q) + f'''(q'))\Big) \end{aligned}$$

As $h \to 0$, $\Big(f'''(q) + f'''(q')\Big) \to 2f'''(x)$, which is just a constant, C. Thus we have

$$= f'(x) + Ch^2$$

where Ch^2 is the error, and so error $= O(h^2)$. This is called *second order convergence* as $h \to 0$. This is the *centered difference algorithm*. A similar algorithm is the *one-sided difference algorithm*, which is *first order convergence*, $O(h)$. The one-sided is given by $f'(x) \approx \frac{f(x+h)-f(x)}{h}$.

5.1.1 Higher Derivatives

Example: Suppose we want to evaluate a second derivative at x.

$$\begin{aligned} f''(x) &= \frac{f'(x+{}^{h}/_{2}) - f'(x-{}^{h}/_{2})}{h} \\ &= \frac{\frac{1}{h}\Big(f(x+h) - f(x)\Big) - \frac{1}{h}\Big(f(x) - f(x-h)\Big)}{h} \\ &= \frac{f(x+h) - 2f(x) + f(x+h)}{h^2} \end{aligned}$$

This can be shown to be $O(h^2)$ by Taylor expansion. This is a basic example of *finite difference methods* used to solve ODEs/PDEs. **Generally smaller h is better, but we must be wary of rounding error and catastrophic cancellation.** Thus it is not necessarily true that Error$\to 0$ as $h \to 0$.

Example: Suppose we use the centered difference formula and the machine gives us $f(x)(1+\varepsilon)$ for some $|\varepsilon| \leq \varepsilon_{mach}$. Note, this is the best accuracy that we could hope for if we ignore rounding error due to machine subtraction and division by $2h$ as well as $fl(x)$. It is essentially a simplified error model.

$$\frac{f(x+h)(1+\varepsilon_1) - f(x-h)(1+\varepsilon_2)}{2h} = \frac{f(x+h) - f(x-h)}{2h} + \frac{f(x+h)\varepsilon_1 - f(x-h)\varepsilon_2}{2h}$$

Can we bound the error term? Want a simpler expression that is not in terms of $\varepsilon_1, \varepsilon_2$. To assume the worst possible case, we take the largest of the two function values $f(x+h), f(x-h)$ and take the maximum of $\varepsilon_1, \varepsilon_2$ given by ε_{mach}. Thus,

$$\begin{aligned} \text{Error} &\leq max_{x\in\{x-h,x+h\}}|f(x)| \cdot \frac{2\varepsilon_{mach}}{2h} \\ &= C\frac{\varepsilon_{mach}}{h} \\ &= O\Big(\frac{\varepsilon_{mach}}{h}\Big) \end{aligned}$$

Thus we must add the theoretical and numerical error,

$$\text{Error} = O(h^2) + O\Big(\frac{\varepsilon_{mach}}{h}\Big)$$

What is the optimal h? The optimal h is one third between back from 0 towards ε_{mach} in logarithmic land. The minimum is roughly where the two types of error balance each other, i.e. are equal,

$$\begin{aligned} h^2 &= \frac{\varepsilon_{mach}}{h} \\ h &= \sqrt[3]{\varepsilon_{mach}} \end{aligned}$$

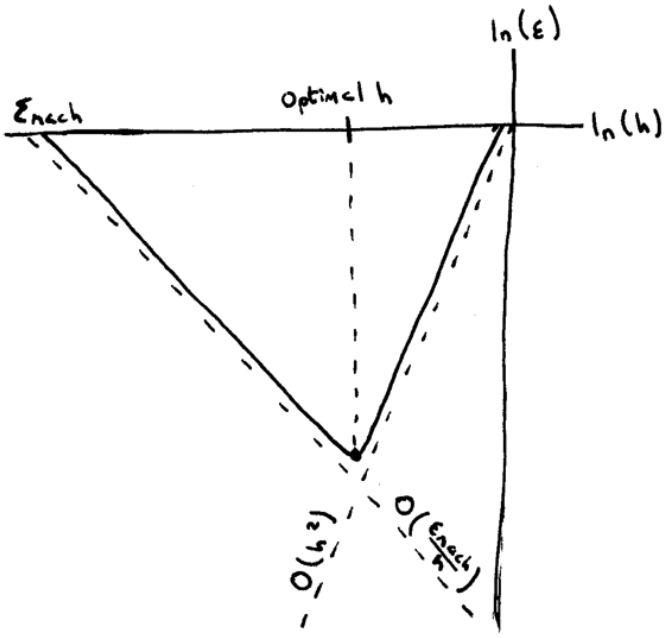

Figure 5.1:

5.2 Stability

Stability is a property of an algorithm, answering the question:

> **Does it solve a problem as accurately as one would expect?**

Def: An algorithm $\hat{f}(x)$ for $f(x)$ is *backwards stable* if $\forall$ inputs x

$$\hat{f}(x) = f\Big(x(1+\varepsilon)\Big) \text{ for some } |\varepsilon| \leq C\varepsilon_{mach}, C < 10^{-3}$$

In words, we can loosely describe this as an algorithm that gives "exactly the right answer to nearly the right question." The strategy for proving backwards stability is to expand $\hat{f}(x)$

Examples: Let us consider backwards stability analyses.

- $f(x) = \sqrt{2}x$.

$$\begin{aligned}\hat{f}(x) &= fl(\sqrt{2}) \times fl(x) \\ &= \sqrt{2}(1+\varepsilon_1) \cdot x(1+\varepsilon_2)(1+\varepsilon_3) \\ &= \sqrt{2}x(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3) \\ &= 1 + (\varepsilon_1+\varepsilon_2+\varepsilon_3) + (\varepsilon_1\varepsilon_2 + ... + \varepsilon_1\varepsilon_2\varepsilon_3) \\ &\leq 1 + 3\varepsilon_{mach}\end{aligned}$$

  Yes it is backwards stable with $|\varepsilon| \leq 3\varepsilon_{mach}$. Note in the last step, we discarded the second portion of error expansion as it was $O(\varepsilon_{mach}^2)$, which was arbitrary compared to $3\varepsilon_{mach}$. The $(1+\varepsilon_3)$ came from the multiplication operation.

- $f(x_1, x_2) = x_1 + x_2$. This is addition with two inputs.

$$\begin{aligned}\hat{f}(x_1, x_2) &= fl(x_1) + fl(x_2) \\ &= \Big((1+\varepsilon_1)x_1 + (1+\varepsilon_2)x_2\Big)(1+\varepsilon_3) \\ &= (1+\varepsilon_1)(1+\varepsilon_3)x_1 + (1+\varepsilon_2)(1+\varepsilon_3)x_2 \\ &= (1+\varepsilon_4)x_1 + (1+\varepsilon_5)x_2, \text{ where } |\varepsilon_4|, |\varepsilon_5| \leq 2\varepsilon_{mach}\end{aligned}$$

  try to write in form $f(x_1(1+\varepsilon_1), x_2(1+\varepsilon_2))$ to match definition of backwards stable.

- Is our algorithm $\sum_{k=0}^{\infty} \frac{1}{k^4}$ backwards stable? Trick question! There is no x input, and to consider backwards stability we must have some function $f(x)$.

There do exist algorithms which are not backwards-stable. A classic example is finding the eigenvalues of a matrix. You write out the characteristic equation and find the roots, which are the eigenvalues. This in fact is a bad algorithm, and not backwards stable. By evaluating characteristic polynomial coefficients, you lose a lot of accuracy.

6 Lecture 6

Let $\tilde{f}(x)$ be a backwards stable algorithm for a problem $f(x)$. How accurate is it?

Theorem: If an algorithm $\tilde{f}(x)$ is backwards stable, then the relative error (also called *forward error*) is given by

$$\left|\frac{\tilde{f}(x) - f(x)}{f(x)}\right| = \kappa(x) O(\varepsilon_{mach})$$

This is bounding our relative error by Error $\leq C\kappa(x)\varepsilon_{mach}$, where C is some constant that we hope is small.

Proof. Let $\tilde{x} = (1+\varepsilon)x$, where $\varepsilon = O(\varepsilon_{mach})$, be tweaked s.t. $\tilde{f}(x) = f(\tilde{x})$. Then $|\delta x| = |\tilde{x} - x| = |\varepsilon x| \leq |x| O(\varepsilon_{mach})$. Now recall the definition of $\kappa(x) = \frac{|\delta f/f|}{|\delta x/x|}$. Combining these definitions gives us $\left|\frac{\delta f}{f}\right| = \kappa(x)\left|\frac{\delta x}{x}\right| = \kappa(x) O(\varepsilon_{mach})$. □

Examples:

- Consider some function f, where $\kappa = 10^3$ with a backwards stable algorithm. Then we are guaranteed $\approx 10^3 \cdot 10^{-16} = 10^{-13}$ accuracy. I.e. we have lost approximately $\log_{10} \kappa(x)$ digits of accuracy.
- Let $f(x) = \sin(10^{10})$, where $\kappa(x) = |x \cot(x)| \approx 2 \cdot 10^{10}$ at $x = 10^{10}$. Thus we can predict a relative error of $\approx 2 \cdot 10^{10} \cdot 10^{-16} = 2 \cdot 10^{-6}$. Calculating our error explicitly using the backwards stable definition, we get

$$\begin{aligned}
\hat{\sin}(10^{10}) &= \sin\left(10^{10}(1+\varepsilon)\right) \\
&= \sin(10^{10} + 10^{10}\varepsilon) \\
&= \sin(10^{10} + 10^{-6}) \\
&= \sin(10^{10})\cos(10^{-6}) + \cos(10^{10})\sin(10^{-6})
\end{aligned}$$

We can then calculate the relative error as

$$\begin{aligned}
\text{Error} &= \left|\frac{\sin(10^{10})\cos(10^{-6}) + \cos(10^{10})\sin(10^{-6}) - \sin(10^{10})}{\sin(10^{10})}\right| \\
&= \left|\cos(10^{-6}) + \frac{\cos(10^{10})}{\sin(10^{10})}\sin(10^{-6}) - 1\right| \\
&= \left|\cos(10^{-6}) + \cot(10^{10})\sin(10^{-6}) - 1\right| \\
&= 1.790992975481132e-06
\end{aligned}$$

This confirms our initial prediction of $2 \cdot 10^{-6}$ bounding the error.

Def: An algorithm is *stable* if

$$\left|\frac{\tilde{f}(x) - f(\tilde{x})}{f(\tilde{x})}\right| = O(\varepsilon_{mach}),$$

for some $\tilde{x} = x(1+\varepsilon), \varepsilon = O(\varepsilon_{mach})$. In words, we can say that the algorithm gives nearly the right answer to nearly the right question. Note, this is a weaker notion than backwards stability

- Addition, subtraction, multiplication and division are all backwards stable.
- $\sin, \cos$, and most functions are only stable.
- Some functions are unstable, e.g. $\sqrt{1+x^2} - 1$ for x close to 0.

6.1 Stability of Linear Systems

Def: The *2-norm* of a vector is its size $||x|| = \sqrt{x_1^2 + ... + x_n^2} = \sqrt{\vec{x} \cdot \vec{x}}$. The error in a vector result is the size of the error vector, $||\hat{x} - x||$. The error of a matrix $||A||$ is defined as the largest factor by which the matrix can grow a vector's length by, called the *induced error*, where

$$\|A\| = \max_{\vec{x} \in \mathbb{R}^n, \vec{x} \neq \vec{0}} \frac{\|Ax\|}{\|x\|}$$

We must generalize the condition number of a matrix-vector multiplication, $\vec{f}(\vec{x}) = A\vec{x}$. First let us note that given a square matrix, $M = N$, then $\frac{\|\vec{x}\|}{\|A\vec{x}\|} \leq \|A^{-1}\|$.

Proof.

$$\|\vec{x}\| = \|A^{-1}A\vec{x}\| \leq \|A^{-1}\| \cdot \|A\vec{x}\|$$

□

Thus let

$$\begin{aligned}
\kappa(A) &= \max_{\delta\vec{x} \in \mathbb{R}^n, \delta\vec{x} \neq \vec{0}} \frac{\|\delta\vec{f}\|/\|\vec{f}\|}{\|\delta\vec{x}\|/\|\vec{x}\|} \\
&= \max \frac{\|\delta\vec{f}\|}{\|\delta\vec{x}\|} \cdot \frac{\|\vec{x}\|}{\|\vec{f}\|} \\
&= \max \frac{\|A\delta\vec{x}\|}{\|\delta\vec{x}\|} \cdot \frac{\|\vec{x}\|}{\|A\vec{x}\|} \\
&\leq \max \|A\| \frac{\|A^{-1}\|\|A\vec{x}\|}{\|A\vec{x}\|} \\
&= \|A\| \cdot \|A^{-1}\|
\end{aligned}$$

The matrix condition number is giving us the worst case relative sensitivity over all tweak directions in 2-norms. Figure 6.1 shows a matrix A mapping the unit circle from $\mathbb{R}^m \to \mathbb{R}^n$. The 2-norm of A is the point mapped furthest, shown on the right plot.

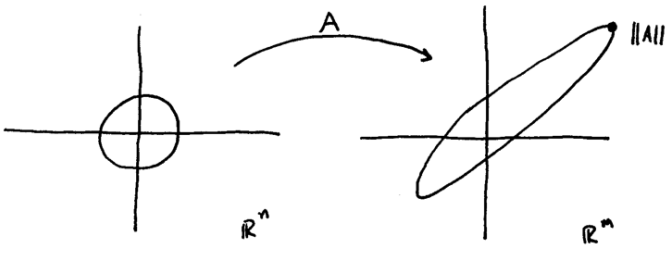

Figure 6.1:

Given a matrix A, MATLAB's $cond(A)$ computes the condition number. Also note that libraries (in Matlab, Sage, etc.) for linear algebra are all backwards stable. Thus the digits lost in $\hat{x} \approx \log_{10} \kappa(A)$, and the relative error is given by $\left|\frac{\hat{x}-\vec{x}}{\vec{x}}\right| = \kappa(A)O(\varepsilon_{mach})$.

Def: The *residual* is $\vec{r} = A\vec{x} - \vec{b}$. Note, $\|\vec{r}\|$ is always small.