MATH 387 CLASS NOTES

McGill university

nicholas hayek

Based on lectures by Prof. Gantumur Tsogtgerel

CONTENTS

I Introduction

APPLIED COMPUTER ARITHMETIC

Integers

We break up the integer 135 as follows: $135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$, and, generally

$$n = \pm \sum_{k}^{\infty} m_k \beta^k \quad \beta \in \mathbb{N}$$

One calls β the *base* or *radix*, and by necessity $0 \le m_k < \beta$. We also see that $\sum m_k$ must be finite, or else the whole sum (i.e. n) is infinity.

From a computer's POV, $\beta = 2$. This representation is called *binary*, and a single binary digit is referred to as a *bit*. In particular, our grade-school algorithm for addition carries over to the binary system:

And 8 bits make a *byte*

$$\begin{array}{cccc} & 1 & 1 & \\ + & 1 & 0 & 1 \\ & & 1 & 1 \\ \hline 1 & 0 & 0 & 0 \end{array} \quad \text{which agrees with } 5 + 3 = 8 =_2 1000$$

Thus, if our thinking machine can add two bits, then it can add any two numbers, up to hardware limitations. Logically, adding two bits is equivalent to an XOR (exclusive or) gate, where $1 + 1$ and $0 + 0$ are not true ($= 0$) while $1 + 0$ and $0 + 1$ *are* true ($= 1$). The carry operation is an AND gate, i.e. $1 + 1$ carries.

A typical modern CPU performs operations on either 32 or 64 bit integers, at the most basic level. One figures, by uniqueness, that a 64-bit number may represent any $n \in [0, 2^{64} - 1]$, which has an upper limit of roughly 1.8×10^19, unsigned. If we desire a *signed* range, i.e. specifying $\pm$, then this range is reduced by half, to $\approx 9 \times 10^{18}$.

Floating Point and Normalization

Any fractional number may be represented as $x = m\beta^e$, where $m \in M$ is called the *mantissa*, β is once again the base, and $e \in E$ is some integer exponent. For example, $x = 2147 \cdot 10^{-2} = 21.47$.

Certain standards exist for representing float point numbers: *IEEE 754 double precision* specifies that e must be an 11 bit, signed number, yielding $e \in [-1022, 1023]$, and m is some 52 bit number between 1 and 2, i.e. $1.m_{-1}m_{-2}...m_{-52}$ (engineers call this normalized). In total, one sees that this float-point number uses 63 bits, in addition to a sign, yielding 64.

Example: Consider $a.bc \cdot 10^e$, where $-1 \leq e \leq 1$, $b, c \in [0, 9]$, and $a \in [1, 9]$. What is the smallest number of this form? Clearly, this is $1.00 \cdot 10^{-1} = 0.1$. The next highest is 0.101, and after that 0.102 (one can check). The largest number such that $e = -1$ is 0.999.

When one lets $e = 0$, we can find the number immediately greater than 0.999, which is $1.00 \cdot 10^0 = 1.00$. Exhausting our list with $e = 0$ leaves us at 9.99. Similarly, when $e = 1$, we graduate one order of magnitude, and find the next largest number to be 10.0, and the absolute largest 99.9.

Consider the precision of this counting exercise: for numbers between .1 and .999, where $e = -1$, our representation is capable of precision on the order of $p = 0.001$. For numbers between 1.00 and 9.99, with $e = 0$, $p = .01$, and, as expected, $p = .1$ for $e = 1$.

We are interested now in the *gap* between 0 and the smallest number one can represent with float-point arithmetic. In the above example, we saw this gap to be 0.1. How about in IEEE 754? We let e be minimal, i.e. $e = -1022$, and our normalized $m := 1.0...0$, so $x_0 = 1.0...0 \times 2^{-1022} = 2^{-1022}$.

This is quite small, but not with respect to the resolution of small numbers in IEEE 754: $x_1 = 1.0...1 \times 2^{-1022} = 2^{-1022} + 2^{-52}2^{-1022}$, and thus $x_1 - x_0 = 2^{-52}2^{-1022} = 2^{-1074}$, which is *52 times* smaller than the gap we observed.

We have a casual notion of normalization so far, and one casual characteristic of normalized mantissa are that they produce these large gaps around zero. Note that our base-10 exercise, too, is structured similarly to IEEE 754, i.e. m is of 1 order of magnitude and its first digit is non-zero.

Numbers which are too large to be represented with a particular amount of bits are contained in *overflow*. One calls the region of numbers too small to be represented, in the case of normalized mantissa, *underflow*.

Theory of Float Point

In practice, one can always represent $x \in \mathbb{R}$ in float point, up to an error of machine insignificance on the order of x (the "relative error" is small).

Axiom 0: For any $x \in \mathbb{R}$, $\exists \tilde{x} \in \tilde{\mathbb{R}} := \{m\beta^e \text{ s.t. } m \in M, e \in \mathbb{Z}\}$ such that $|\tilde{x} - x| \leq \varepsilon|x|$, where one considers $\varepsilon \approx 10^{-16}$, called "machine epsilon."

Proposition 1.1 0 is represented exactly

Proof. We cannot assume that $0 \in M$, or else the result would be automatic. Thus, fix $\tilde{x} \neq 0$ to be such that $\tilde{x} \leq x \in \tilde{\mathbb{R}}$ for any x. We have that $|\tilde{x}| \leq \varepsilon|0| = 0 \implies |\tilde{x}| \leq 0$, and thus $\tilde{x} = 0$.

Now, is this $\tilde{x}$ guaranteed to exist? Should we be looking for the infimum instead? Is analysis being thrown out the window? All yes, but we won't do anything about that. □

In theory, we cannot assume that 1 is represented exactly. Furthermore, we cannot assume that float-point representations are always closed under arithmetic (+, -, ×, ÷). For example, suppose we restrict the mantissa to be 3 digits, and consider $1.00 \times 10^4 + 1.00 = 10,001$. This result requires a 5 digit mantissa, and so float point numbers, generally, aren't closed under arithmetic.

Axiom 1: Choose an operation $* \in \{+,-,\times,\div\}$. Then the operation $\tilde{\mathbb{R}} \times \tilde{\mathbb{R}} \mapsto \tilde{\mathbb{R}}$ sending $(x,y) \mapsto x \circledast y$, where $\circledast$ is an approximation of $*$, satisfies the following:

$$|x \circledast y - x * y| \leq \varepsilon |x * y|$$

Example: Suppose our mantissa has 3 digits, and consider the equation $10.1 - 9.95 = 0.15$. This is an exact answer. However, we will need 4 digits of representation to properly subtract these two numbers (10.10 and 09.95), so one throws out the 5 in 9.95 and considers the subtraction of 10.1 and 09.9. $x \circledast y = .2$ while $x * y = .15$, and we conclude an error of .05.

II Error Analysis

ROUND-OFF ERROR

For a value x and its approximation $\tilde{x}$, define the *absolute error* of $\tilde{x}$ to be $|x - \tilde{x}|$. Define the *relative error* of $\tilde{x}$ to be $\frac{|x-\tilde{x}|}{x}$. As the name suggests, relative error will scale with the input x, and will be the pertinent measure of error in this course.

From Axiom 0, we have that, for any $x \in \mathbb{R}$, there exists some $\tilde{x}$ such that $|\frac{\tilde{x}-x}{x}| \leq \varepsilon$, or, equivalently, $|\frac{\tilde{x}-x}{x}| = \delta$, where $|\delta| \leq \varepsilon$. Rearranging, one yields $\tilde{x} = x(1+\delta)$. This form will be used frequently in error analysis. Proposition 1.1

Note that $\tilde{x}$ and x may be $a \circledast b$ and $a * b$, respectively, according to Axiom 1. The useful form $\tilde{x} = x(1+\delta)$ still holds.

Example: Let $z = a + b - c$, where a, b, c are represented exactly (without input error), and let $\tilde{z} = a \oplus b \ominus c$. Assume we are performing "naive" addition/subtraction, i.e. computing $a \oplus b$, then subtracting from that c.

$\implies a \oplus b \ominus c = (a+b)(1+\delta_1) \ominus c = [(a+b)(1+\delta_1) - c](1+\delta_2)$, where $|\delta_1|, |\delta_2| \leq \varepsilon$. One then concludes $\tilde{z} - z = (a+b)(\delta_1 + \delta_2 + \delta_1\delta_2) - c\delta_2$.

$$\begin{aligned} |\tilde{z} - z| &\leq |a+b|(|\delta_1| + |\delta_2| + |\delta_1||\delta_2| - |c||\delta_2|) \\ &\leq |a+b|(2\varepsilon + \varepsilon^2) - |c|\varepsilon = (2|a+b| + |c|)\varepsilon + |a+b|\varepsilon^2 \end{aligned}$$

In practice, one may ignore the ε^2 term, and write $|\tilde{z} - z| = (2|a+b| + |c|)\varepsilon$. Then

our relative error is
$$\frac{|\tilde{z}-z|}{|z|} = \frac{2|a+b|+|c|}{|a+b-c|}\varepsilon + o(\varepsilon^2)$$

Propagation of Error

Whereas above we considered exact inputs and in-exact operations, here we will consider *exact* operation, but inexact inputs. This is called propagation of error," and quantifies the effect that perturbations of input have on an operation or function. We follow from the same axioms and definitions, so to lead off, an example:

Example: Let $z = a + b$ and $\tilde{z} = \tilde{a} + \tilde{b}$. Then $\frac{\Delta z}{z} = \frac{\Delta a + \Delta b}{a+b}$, where $\Delta z = \tilde{z} - z$, and similarly for a and b. Rearranging, one writes

$$\frac{\Delta z}{z} = \frac{1}{a+b}\left(a\cancelto{\varepsilon_a}{\frac{\Delta a}{a}} + b\cancelto{\varepsilon_b}{\frac{\Delta b}{b}}\right)$$

This will be convenient for us, since our relative error $\frac{\Delta z}{z}$ is represented in terms of the relative errors of a and b. Let $\varepsilon_a \approx \varepsilon_b = \varepsilon$ be these (approximately equal, by assumption) errors. We have, finally

$$\left|\frac{\Delta z}{z}\right| = \frac{|a\varepsilon_a + b\varepsilon_b|}{|a+b|} \le \frac{|a|+|b|}{|a+b|}|\varepsilon|$$

This last term, $\frac{|a|+|b|}{|a+b|}$, is called the *condition number* for addition. In fact, all differentiable operations have such a condition number.

1.1 Condition Number for Arbitrary Function

Let $y = f(x)$ be a differentiable function. Then

$$\frac{\Delta y}{y} = \frac{xf'(x)}{f(x)}\varepsilon_x \quad \text{where } \varepsilon_x \text{ is the relative error of our input}$$

Furthermore, we call $\kappa = \frac{xf'(x)}{f(x)}$ the *condition number* for f

---

Examples: To get a feel for this, we'll compute a few condition numbers:

1. $f = e^x$, then $\kappa = \frac{xe^x}{e^x} = x$.

2. $f = \ln(x)$, then $\kappa = \frac{x\frac{1}{x}}{\ln(x)} = \frac{1}{\ln(x)}$

3. $f = x - 1$, then $\kappa = 1 + \frac{1}{x-1}$. This result, in particular, is intuitive: it will be incredibly difficult to represent $x - 1$ when $x \to 1$, and so our relative error balloons around 1.

1.2 Multivariate Condition Number

If $z = f(x, y)$ is a function of two inputs, and the relative errors ε_x, ε_y of x and y, respectively, are roughly equivalent, the condition number κ for z satisfies $\frac{\Delta z}{z} = \kappa\varepsilon$, where $\varepsilon_x \approx \varepsilon_y = \varepsilon$. Then, κ is given by

$$\kappa = \left|\frac{x}{z}\frac{\partial z}{\partial x}\right| + \left|\frac{y}{z}\frac{\partial z}{\partial y}\right| = \left|\frac{x f_x}{f}\right| = \left|\frac{y f_y}{f}\right|$$

This generalizes easily to n inputs.

Examples: Once again, let's calculate the condition number for some multivariate functions:

1. $f(x, y) = x \pm y \implies f_x = 1$ and $f_y = \pm 1$, so $\kappa = \frac{|x|+|y|}{|x \pm y|}$

See the example from the top of p. 6

2. $f(x, y) = xy \implies f_x = y$ and $f_y = x$, so $\kappa = 2$
3. $f(x, y) = \frac{x}{y} \implies f_x = \frac{1}{y}$ and $f_y = \frac{-x}{y^2}$, so $\kappa = 2$

i.e. multiplication and division preserve the magnitude of error in their inputs

4. $s = x_1 + ... + x_n$ will yield $\kappa = \frac{|x_1|+...+|x_n|}{|x_1+...+x_n|}$ and $p = x_1...x_n$ will yield $\kappa = n$. These are particularly useful results.

NAIVE ALGORITHMS

Naive Multiplication

With the tools we've developed thus-far, we'll consider the error inherent to *naive multiplication*, a process described in the picture below:

$$\begin{tikzcd}
x_1 \arrow[r] & \otimes \arrow[r] & \otimes \arrow[r] & \otimes \arrow[r, dashed, no head] & {} \\
x_2 \arrow[ur] & x_3 \arrow[ur] & x_4 \arrow[ur] & &
\end{tikzcd}$$

First we'll analyze the round-off error from multiplication. Let $p_n = x_1 \times x_2 \times ... \times x_n$ and $\tilde{p_n} = x_1 \otimes x_2 \otimes ... \otimes x_n$, where $\otimes$ is inexact. Then, from proposition 1.1, we have $\tilde{p_n} = x_1 x_2 ... x_n (1+\delta_1)(1+\delta_2)...(1+\delta_n)$, where $|\delta_i| \le \varepsilon$. This is exactly $p_n(1+\delta_1)...(1+\delta_n) = p_n + p_n(D(\delta))$, where $D(\delta) = \delta_1 + ... + \delta_n + \delta_1\delta_2 + ... + \delta_1\delta_2\delta_3 + ... + \delta_1...\delta_n$, i.e. all unique combinations of $\delta_1...\delta_i$, where i ranges from 1 to n. Since $|\delta_i| \le \varepsilon$, we can state a convenient inequality, using basic combinatorics:

$$\begin{aligned}
\tilde{p_n} - p_n = p_n D(\delta) \implies \frac{\tilde{p_n} - p_n}{p_n} &= D(\delta) && (1)\\
&\le \binom{n-1}{1}\varepsilon + \binom{n-1}{2}\varepsilon^2 + ... + \binom{n-1}{n-1}\varepsilon^{n-1} && (2)\\
&\le (n-1)\varepsilon + (n-1)^2\varepsilon^2 + ... + (n-1)^{n-1}\varepsilon^{n-1} && (3)\\
&\le \frac{(n-1)\varepsilon}{1-(n-1)\varepsilon} \le \frac{n\varepsilon}{1-n\varepsilon} && (4)
\end{aligned}$$

Step (3) follows from a rough bound on the factorial: $\binom{n-1}{k} = \frac{(n-1)!}{k!(n-1-k)!} \le (n-k)(n-k-1)...(n-1) \le (n-1)^k$, and (4) is just a geometric series. If we assume that $n\varepsilon \le \frac{1}{2}$ (which is fair, given that $\varepsilon \approx 10^{-16}$, and one would need to compute about 5 quadrillion consecutive naive multiplications to break this bound), we can conclude

$$\frac{n\varepsilon}{1-n\varepsilon} \le 2n\varepsilon$$

We'll now move on the case where x_i are *not* necessarily exact. *Re*define $\tilde{p_n} = \tilde{x_1}...\tilde{x_n}$, and let $\hat{p_n} = \tilde{x_1} \otimes ... \otimes \tilde{x_n}$ and p_n be as above. We already know that $\frac{\tilde{p_n}-\hat{p_n}}{\tilde{p_n}} \le 2n\varepsilon$, and we *want* to consider $\frac{\hat{p_n}-p_n}{p_n}$:

$$\begin{aligned}
|\hat{p_n} - p_n| = |\hat{p_n} - \tilde{p_n} + \tilde{p_n} - p_n| &\le |\hat{p_n} - \tilde{p_n}| + |\tilde{p_n} - p_n|\\
&\le 2n\varepsilon|\tilde{p_n}| + n\left|\frac{\Delta x}{x}\right||p_n| \le 2n\varepsilon|\tilde{p_n}| + n\delta|p_n|
\end{aligned}$$

Asymptotically, when $\delta \to 0$, $\tilde{p_n} \approx p_n$, so we can bound by $n(2\varepsilon + \delta)|p_n|$

Naive Summation

We'll do exactly what we did above, but for naive summations:

$$\begin{tikzcd} x_1 \arrow[r] & \oplus \arrow[r] & \oplus \arrow[r] & \oplus \arrow[r, dashed, no head] & {} \\ x_2 \arrow[ur] & x_3 \arrow[ur] & x_4 \arrow[ur] & & \end{tikzcd}$$

Consider $z_n = x_1 + ... + x_n$ and $\hat{z_n} = x_1 \oplus ... \oplus x_n$, where the inputs are exact. We observe the following:

$$\begin{aligned} \hat{z_2} &= (x_1 + x_2)(1 + \delta_1) \\ \hat{z_3} &= [(x_1 + x_2)(1 + \delta_1) + x_3](1 + \delta_2) \qquad\qquad |\delta_i| \leq \varepsilon \\ \hat{z_4} &= [(x_1 + x_2)(1 + \delta_1)(1 + \delta_2) + x_3(1 + \delta_2) + x_4](1 + \delta_3) \\ &\vdots \\ \hat{z_n} &= (x_1 + x_2)(1 + \delta_1)...(1 + \delta_{n-1}) + x_3(1 + \delta_2)...(1 + \delta_{n-1}) + ... + x_n(1 + \delta_{n-1}) \\ \implies |\hat{z_n} - z_n| &\leq (|x_1| + |x_2|)\frac{n\varepsilon}{1 - n\varepsilon} + |x_3|\frac{(n-1)\varepsilon}{1 - (n-1)\varepsilon} + ... + |x_{n-1}|\frac{2\varepsilon}{1 - 2\varepsilon} + |x_n|\varepsilon \end{aligned}$$

If $n\varepsilon \leq \frac{1}{2}$, then $\leq 2n\varepsilon(|x_1| + |x_2|) + 2(n-1)\varepsilon|x_3| + ... + 4\varepsilon|x_{n-1}| + \varepsilon|x_n|$

$\leq 2n\varepsilon(|x_1| + |x_2| + ... + |x_n|)$, where $n \geq 2$

Finally, we get

$$\frac{|\hat{z_n} - z_n|}{|z_n|} \leq 2n\left(\frac{|x_1| + ... + |x_n|}{|x_1 + ... + x_n|}\right)\varepsilon$$

III Elementary Functions

Argument Reduction

Suppose we want to compute $\sqrt{x}$, $\sqrt[3]{x}$, e^x, or some other analytic function. We are already equipped with Taylor series from previous calculus courses, but these are sometimes bad when inputs are big. Thus, we perform *argument reduction* to make our inputs small. This is completely dependent on the function we are analyzing, but consider e^x:

We know that $e^x = \left(e^{x/n}\right)^n$, and $e^x \approx 1 + x$ when $x \approx 0$. The following program will shrink our inputs such that this Taylor approximation is OK, for sufficient $\varepsilon > 0$.

```
def eulerSeries(x, e):
  index = 0
  while x > e:
    x = x/2
    index += 1

  return (1+x)**(2**n)

```