Notes for An Introduction to Mathematical Cryptography

Mohamed-Amine Azzouz

2024-05-03

Contents

Lecture 1: An Introduction to Cryptography

Introduction

Note: these are a set of notes to help me summarize my casual reading of the textbook of the same name.
Nothing noted yet

Preliminary to 2.1 Well-posedness and Condition Number of a Problem

- A numerical problem is well-posed if a small change in the parameters given only results in a small change in the result. Otherwise, it is ill-posed.
- This idea is synonymous to numerical stability, which can be quantified with the relative and absolute condition numbers $K(d)$, where d is the data used by the numerical method.

Lecture 6 will expand on those notions.

2.2 Stability of Numerical Methods

Assume the numerical problem we are to solve is well-posed. Oftentimes, we want to simplify the problem posed into a sequence of simpler, but approximate problems.

- A method, which is a sequence of approximate problems, is consistent if that sequence converges to the original numerical problem.
- A method is strongly consistent if each approximate problem gives the same solution as the original problem given the same data.

To measure the convergence of a method (which yields the result x_n to the true result x, we use metrics such as relative and absolute error:

Definition 1.1: Relative and absolute error

- **Absolute error** between x_n and x is defined by

$$\varepsilon = E_{\text{abs}}(x_n) = |x - x_n|$$

- If $x \neq 0$, the **relative error** between x_n and x is defined by

$$\eta = E_{\text{rel}}(x_n) = \frac{|x - x_n|}{|x|}$$

Remark 1.1

The definition above also works whenever x_n and x live in normed vector spaces, where

the absolute values are replaced by the appropriate norm.
ASIDE: In the case where x_n and x are vectors or matrices, we can think of an component-wise form of error. In that case, if $x \neq 0$, the **relative error by component** x is defined by

$$E^c_{\text{rel}}(x_n) = \max_{i,j} \frac{|(x - x_n)_{ij}|}{|x_{ij}|}.$$

One last thing about stability and convergence of a numerical is that they are equivalent under certain conditions. It can be shown that:

Proposition 1.1: Lax-Richtmyer Equivalence Theorem

For a consistent numerical method, stability is equivalent to convergence.

Preliminary to 2.3 A priori and a posteriori Analysis

They are means of evaluating the error induced by a numerical method. This will be expanded upon later on Lecture 6.

2.4 Sources of Error in Computational Models

Oftentimes, a numerical problem approximates a mathematical problem, which in terms is a model of a physical problem. We can call it a computational model for said physical problem.

- The **global error** is the difference between the computed solution and the physical solution;
- The global error is the sum of the **mathematical model** and the **computational model**.

2.5 Machine Representation of Numbers

Let $\beta \in \mathbb{N}$ be the base of a number. We need to have $\beta \geq 2$.
First, we shall review representations of integers, mostly focusing on binary:

Definition 1.2: Unsigned integers

Let $x \in \{0\} \cup \mathbb{N}$. We can then represent it as this sum of n terms:

$$x_\beta = \sum_{k=0}^{n-1} x_k \beta^k,$$

where $0 \leq x_k < \beta$ for any $0 \leq k < n$. Note that n entries allow us to represent any integer in $[0, \beta^n - 1]$.

Definition 1.3: Signed integers: sign and magnitude

Let $x \in \mathbb{Z}$. We can then represent it as this sum of n terms and one sign bit s as:

$$x_\beta = (-1)^s \sum_{k=0}^{n-1} x_k \beta^k,$$

where $0 \leq x_k < \beta$ for any $0 \leq k < n$. s is equal to 0 if the number is positive, 1 if it is negative. Note that n entries allow us to represent any integer in $[-\beta^n - 1, \beta^n - 1]$, with 0 having two representations.

However, as you may have seen in ECSE 222 and ECSE 324, the following method of representing signed integers is usually preferred since it makes arithmetic on digital circuits and processors simpler to implement:

Definition 1.4: Signed integers: 2's complement

Let $\beta = 2$, $x \in \mathbb{Z}$. We can then represent x as this sum of n terms and :

$$x = [x_{n-1} x_{n-2} \dots x_1 x_0]$$

where $x_k \in \{0, 1\}$ for any $0 \leq k < n$. Note that n bits allow us to represent any integer in $[-2^n, 2^{n-1} - 1]$. If $x_{n-1} = 0$, x is positive, and $\sum_{k=0}^{n-2} x_k 2^k$. If $x_{n-1} = 1$, it is negative. To find $-x$ from x, flip all of x's bits and add 1.

In general, we can now write down any real number x with finitely many digits as such:

Definition 1.5: Positional system

If x has finitely many digits, one may write it as

$$x_\beta = (-1)^s [x_n x_{n-1} \dots x_1 x_0 . x_{-1} x_{-2} \dots x_{-m}],$$

where $x_m \neq 0$, and $0 \leq x_k < \beta$ for any $-m \leq k \leq n$. s is equal to 0 if the number is positive, 1 if it is negative. In fact, this representation is equivalent to:

$$x_\beta = (-1)^s \sum_{k=-m}^{n} x_k \beta^k.$$

Remark 1.2

For any base β, it is easy to show that for any $\varepsilon > 0$, and $x_\beta \in \mathbb{R}$, there exists $y_\beta \in \mathbb{R}$ with finitely many digits such that $|y_\beta - x_\beta| < \varepsilon$.

This representation, when $\beta = 2$, is called the **fixed-point system**. Definition 1.2 can be rewritten as:

$$x_\beta = (-1)^s [a_{N-2} a_{N-3} \dots a_k . a_{k-1} \dots a_0] = (-1)^s \beta^{-k} \sum_{j=0}^{N-2} a_j \beta^j,$$

giving us N total values: one for the sign bit, k for the fractional part, and $N - k - 1$ for the integer part.
Drawback of fixed point: strongly limits the value of the minimum and maximum numbers that can be represented, unless N is very large.
To fix this, the exponential scaling k can be allowed to be varying as such:

Definition 1.6: Floating-point representation

If $x \neq 0$ has finitely many digits, one may write it as

$$x_\beta = (-1)^s \beta^e [0.a_{t-1} \dots a_1 a_0] = (-1)^s \beta^{e-t} m,$$

where:

- $m = [a_{t-1} \dots a_1 a_0]$ is the **mantissa**, with $0 \leq m \leq \beta^t - 1$;
- t is the number of significant figures a_i, with $0 \leq a_i \leq \beta - 1$;
- e is the exponent, or the number of digits in the integer part.

This is the number's **floating-point representation**. Notice that the size of the mantissa is equal to t, unless binary is used. In that case, normalization forces a_{t-1} to 1, making the size of the mantissa to be $t - 1$.

Therefore, in terms of computer storage, the N entries (bits when $\beta = 2$) are stored like so:

- 1 bit for the sign s;
- t entries for the mantissa ($t - 1$ if binary);
- The remaining $N - 1 - t$ entries to store the exponent ($N - t$ if binary).

Note that the exponent can only vary between two values $L \leq e \leq U$, where usually $L < 0$ and $U > 0$. This notation is similar to the scientific notation, since we keep track of the significant digits in the mantissa, and exponent gives an idea on the number's scale.

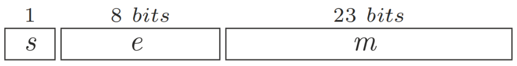

Figure 1: Single precision 32 bit representation.

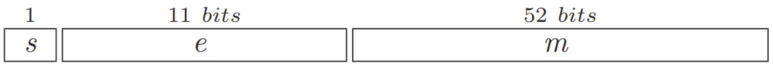

Figure 2: Double precision 64 bit representation.

Typically, computers work with two formats for floating-point: single and double precision. In binary ($\beta = 2$), we tend to see:

- The **float** data type, of single precision, where $N = 32$ bits, it can be graphically represented as in Figure 1 above;
- The **double** data type, of double precision, where $N = 64$ bits, it can be graphically represented as in Figure 2 above.

Definition 1.7: Set of floating-point numbers

Define the set of floating-points numbers with t significant figures, base $\beta \geq 2$, $0 \leq a_i \leq \beta - 1$, and exponent range (L, U) with $L \leq e \leq U$ to be

$$\mathbb{F}(\beta, t, L, U) := \{0\} \cup \{x \in \mathbb{R} : x = (-1)^s \beta^{e-t} \sum_{i=0}^{t-1} a_i \beta^i\}.$$

This representation taken at face value gives us the set of **denormalized floating-point numbers**. To make sure each number has a unique representation, we assume that $a_{t-1} \neq 0$, or that $m \geq \beta^{t-1}$. In that case, a_{t-1} is the principal significant digit (always equal to 1 in binary representations), and we end up with the set of **normalized floating-point numbers**, with $\beta^{t-1} \leq m \leq \beta^t - 1$.

Remark 1.3

Just like in sign and magnitude representations, 0 cannot be represented uniquely, since the bit sign s does not affect 0's value. One tends to assume 0 is positive (in other words, assume the unique representation of 0 uses $s = 0$), but using both representations can be useful in representing concepts such that 0^+ and 0^-. This notation can be seen in the context of limits. For example,

$$\lim_{x \to 0^+} f(x), \quad \lim_{x \to 0^-} f(x).$$

Next, because of the sign bit, we notice that $-x \in \mathbb{F}(\beta, t, L, U)$ if and only if $x \in \mathbb{F}(\beta, t, L, U)$. Furthermore, for normalized values, we notice that the values of x are in the range:

$$x_{min} = \beta^{L-1} \leq |x| \leq \beta^{U}(U - L + 1) + 1 = x_{max}.$$

While the number of possible normalized values is the cardinality:

$$|\mathbb{F}(\beta, t, L, U)| = 2(\beta - 1)\beta^{t-1}(U - L + 1) + 1.$$

To represent numbers smaller than x_{min} in absolute value, one needs to extend $\mathbb{F}$ to the set $\mathbb{F}_D$ of denormalized numbers. If we only do that extension for those small numbers, uniqueness is preserved, while being able to represent numbers whose absolute values are as small as β^{L-t}.
Now, let's look at the distribution of floating-point numbers:

- The absolute distance between two consecutive numbers x and y is $\varepsilon = |x - y| = \beta^{e-t}$.
- However, within any interval $[\beta^{e}, \beta^{e+1}]$, those numbers are equally spaced, but that spacing increases exponentially as e increases.
- The absolute distance between two consecutive numbers 1 and $y > 1$ is $|1 - y| = \beta^{1-t}$, which we call as the **machine epsilon**, denoted ε_m.
- The relative distance between two consecutive numbers x and y is $\eta = \frac{|x-y|}{|x|} = \frac{\beta^{e-t}}{m\beta^{e-t}} = \frac{1}{m}$, where m is the mantissa of x.

We can develop η to give it a range of possible values. First, notice that:

$$\eta = \frac{1}{m} = \frac{1}{a_{t-1} \ldots a_0} = \frac{\beta^{-t}}{(0.a_{t-1} \ldots a_0)}.$$

Since $\beta^{-1}a_{t-1} \leq (0.a_{t-1} \ldots a_0) \leq 1$, it follows that:

$$\beta^{-1}\varepsilon_m = \beta^{-t} = \eta = \frac{|x - y|}{|x|} \leq \beta^{1-t} = \varepsilon_m,$$

so the relative distance only depends on the inverse of the mantissa.
Consequentially, as x increases within $[\beta^{e}, \beta^{e+1}]$, the mantissa increases from β^{t-1} to $\beta^{t} - 1$ (for normalized values), and the relative distance decreases from β^{1-t} to $(\beta^{t} - 1)^{-1} \approx \beta^{t}$. As x reaches β^{e+1}, the relative precision jumps back up to β^{1-t}, as shown on the figure next page. This phenomenon is called **wobbling precision**, and is more pronounced as β increases, so working with binary minimizes this issue.
To get into more practical matters, we should define and discuss the IEC559/IEEE754 standard which is most widely used:

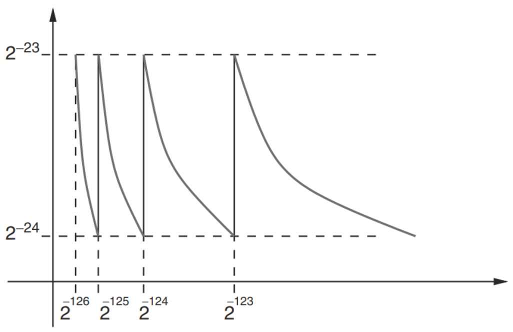

Figure 3: Variation of relative distance for the set of numbers $\mathbb{F}(2, 24, -125, 128)$ IEC559/IEEE754 in single precision.

	$E = 0/e = -2^{N-t-1} + 2$	Other values	$E = 2^{N-t} - 1/e = 2^{N-t-1} + 1$
mantissa $= 0$	± 0	$(-1)^s(0.10)2^e$	$\pm\infty$
mantissa $\neq 0$	denormalized	$(-1)^s(0.1\text{mantissa})2^e$	NaN (not a number)

Definition 1.8: IEC559/IEEE754 standards

Define the following sets of floating-point numbers (according to IEC559/IEEE754):

- quarter() $= \mathbb{F}(2, 5, -1, 4)$ ($N = 8$ bits are used);
- half() $= \mathbb{F}(2, 11, -13, 16)$ ($N = 16$ bits are used);
- single() $=$ float() $= \mathbb{F}(2, 24, -125, 128)$ ($N = 32$ bits are used);
- double() $= \mathbb{F}(2, 53, -1021, 1024)$ ($N = 64$ bits are used).

They respectively denote quarter, half, single, and double precision.

For general binary sets $\mathbb{F}(2, t, L, U)$ stored in N bits, we obtain using E to denote the value actually stored in memory, $t - 1$ to be the size of the mantissa, and e to be the exponent as used so far (i.e. the number of entries in the integer part), we get:

- Number of bits to store $E = N - t$;
- bias $= 2^{N-t-1} - 2$;
- range of $E = [0, 2^{N-t} - 1]$;
- range of $e = [-2^{N-t-1} + 2, 2^{N-t-1} + 1]$.
  Now, we must take two values in the range to take care of the edge cases. Those are, considering that $E = e-$bias:

	$E = 0/e' = -2^{N-t-1}+1$	Other values	$E = 2^{N-t}-1/e' = 2^{N-t-1}$
mantissa $= 0$	± 0	$(-1)^s(1.0)2^{e'}$	$\pm\infty$
mantissa $\neq 0$	denormalized	$(-1)^s(1.\text{mantissa})2^{e'}$	NaN (not a number)

NOTES	N	t-1	N-t	E range	bias	e range	e woe
quarter()	8	4	3	[0,7]	2	[-2,5]	[-1,4]
half()	16	10	5	[0,31]	14	[-14,17]	[-13,16]
single()	32	23	8	[0,255]	126	[-126,129]	[-125,128]
double()	64	52	11	[0,2047]	1022	[-1022,1025]	[-1021,1024]

After removing the edge cases, the range for E is $[1, 2^{N-t}-2]$, and the range of e is $[-2^{N-t-1}+3, 2^{N-t-1}]$, which are good values to give for the lower and upper bound parameters ($L = -2^{N-t-1}+3$, $U = 2^{N-t-1}$).
We can reformulate the same thing in a more similar way to the slides, where we define a new exponent $e' = e - 1$, which is an exponent more closely related to scientific notation. That way, the tables become:

- Number of bits to store $E = N - t$;
- bias' $= 2^{N-t-1} - 1 =$ bias $+1$;
- range of $E = [0, 2^{N-t}-1]$;
- range of $e' = [-2^{N-t-1}+1, 2^{N-t-1}]$.

Now, we must take care of the edge cases, as done in the second table above, with $E = e'-$bias'.
After removing the edge cases, the range for E is $[1, 2^{N-t}-2]$, and the range of e' is $[-2^{N-t-1}+2, 2^{N-t-1}-1]$.
We can plug in these values to produce the third table for the IEEE values (woe means "without edge cases"):
Let's now think of other practical issues with floating-point numbers. The first issue is about the idea of best approximating any $x \in \mathbb{R}$ (as long as it is not so big that it goes outside the range of numbers the system can represent). The second issue is about defining an arithmetic for $\mathbb{F}$, since even if $x, y \in \mathbb{F}$, a binary operation that uses the two won't always yield another element of $x, y \in \mathbb{F}$.

SLIDES	N	t-1	N-t	E range	bias'	e' range	e' woe
quarter()	8	4	3	[0,7]	3	[-3,4]	[-2,3]
half()	16	10	5	[0,31]	15	[-15,16]	[-14,15]
single()	32	23	8	[0,255]	127	[-127,128]	[-126,127]
double()	64	52	11	[0,2047]	1023	[-1023,1024]	[-1022,1023]

To solve the first problem, we must define, for any appropriate $x \in \mathbb{R}$, an operation which outputs the closest floating-point number. For this define the map $fl : \mathbb{R} \to \mathbb{F}$ by

$$fl(x) = (-1)^s(0.a_{t-1}\dots a_1\widetilde{a_0})\beta^e,\ \widetilde{a_0} = \begin{cases} a_0 \text{ if the first cut off term } a_{-1} < \beta/2 \\ a_0 + 1 \text{ if the first cut off term } a_{-1} \geq \beta/2 \end{cases}$$

This mapping is called the **rounding map**. If instead we always have $\widetilde{a_0} = a_0$ it is instead called the **chopping map**.
Some properties of $fl(x)$:

- $fl(x) = x$ for all $x \in \mathbb{R}$;
- If $x \leq y$, then $fl(x) \leq fl(y)$;
- If $x \in \mathbb{R}$, then $x(1-u) \leq fl(x) \leq x(1+u)$, where $u = \frac{1}{2}\beta^{1-t} = \frac{1}{2}\varepsilon_m$ is called the **roundoff unit**.

In other words, the relative error due to rounding becomes:

$$\eta = E_{\text{rel}}(x) = \frac{|x - fl(x)|}{|x|} \leq u$$

Then, by definition of $fl(x)$, the absolute error is:

$$\varepsilon = E(x) = |x - fl(x)| \leq \beta^{e-t}|(a_{t-1}\dots a_1a_0.a_{-1}\dots) - (a_{t-1}\dots a_1\widetilde{a_0})| \leq \frac{1}{2}\beta^{e-t}.$$

The second issue, which is to create an analogous operation from $\circ$ (which could denote an addition, subtraction, multiplication, or division), is to denote an analogous operation $\bullet : \mathbb{R} \times \mathbb{R} \to \mathbb{F}$, to be defined as:

$$x \bullet y := fl(fl(x) \circ fl(y)).$$

This operation commutes for additions and multiplications, but other properties like associativity are lost. Also, for operations like subtractions, if we want to have the typical roundoff precision of $(x \circ y)(1-u) \leq x \bullet y \leq (x \circ y)(1+u)$, we may need to keep track of other variables, such as rounding digits, which most computers do keep track to make sure the result of a subtraction isn't too far off. An arithmetic that does not have that roundoff precision is called aberrant.

Lecture 2: Discrete Logarithms and Diffie–Hellman

XXXXXXXXXXX

XXX

YYY

Lecture 3: Integer Factorization and RSA

XXXXXXXXXXXDDDD

XXX

YYY

Lecture 4: Digital Signatures

:3 XXXXXXXXXXDDDD

XXX

YYY

Lecture 5: Combinatorics, Probability, and Information Theory

:3 XXXXXXXXXXDDDD

XXX

YYY

Lecture 6: Elliptic Curves and Cryptography

:3 X

XXX

YYY

Lecture 7: Lattices and Cryptography

:3 X

XXX

YYY

Lecture 8: Additional Topics in Cryptography

:3 X

XXX

YYY

Bibliography

[1] J. Hoffstein, J. Pipher, and J. H. Silverman, *An Introduction to Mathematical Cryptography*, 2nd ed. New York, NY: Springer, 2014. [Online]. Available: https://doi.org/10.1007/978-1-4939-1711-2