

## Lecture 03: Sources of Error cont'd

### Life in *Floating-Point Land*

#### Outline

- 1) Truncation error vs. Floating Point Error
- 2) Floating Point Systems:
  - Basic definitions
  - Basic Operations
  - Rounding vs. Chopping
- 3) Example: a  $p=1$ ,  $p=2$  toy decimal system
  - Machine epsilon
- 4) IEEE single and double Precision floating point
- 5) The great exp challenge -- truncation & Floating Point error

## Review: Sources of Error

- 1) "Truncation Error": e.g. Errors arising from approximating a function with a simpler function e.g.

$$\sin(x) \approx x - x^3/3! + x^5/5! + \mathbf{O}(h^7)$$

- 2) "Floating Point Error": Errors arising from approximating real numbers with finite-precision numbers e.g.

$$\pi \approx 3.14 \text{ (a 3 digit floating-point systems)}$$

## Floating Point Systems: definitions

All *normalized floating point systems* can be described as the following

$$\hat{f} = \pm d_1.d_2d_3d_4\dots d_p \times \beta^E$$

where:

- 1)  $\pm$  is a sign bit
- 2)  $d_1.d_2\dots d_p$  is the *Mantissa* (and  $.d_2\dots d_p$  is the "*fraction*" with  $p$  digits of precision (normalized FP systems assumes  $d_1 \neq 0$ ))
- 3)  $\beta$  is the *base* (e.g. binary fp  $\beta=2$ , decimal  $\beta=10$ )
- 4)  $E$  is the *exponent*, an *integer* in the range  $[E_{\min}, E_{\max}]$

Comments:

- 1) Floating point systems form a *discrete and finite set* of numbers
- 2) Floating point numbers are *not* uniformly distributed on the real line
- 3) Arithmetic in FP systems is different from real math

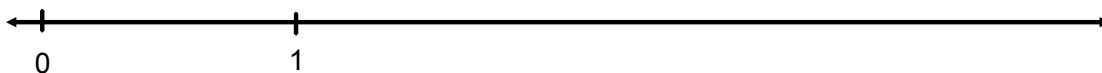
## Example: a Toy system

Consider the toy 2-digit precision decimal systems

$$f = \pm d_1.d_2 \times 10^E$$

with  $E$  in  $[-2, 0]$

- 1) How many numbers in  $f$ ?
- 2) Distribution on the real line



## **Example: a Toy system**

Basic Arithmetic

Multiplication

## **Example: a Toy system**

Basic Arithmetic

Addition

## Example: a Toy system

Basic Arithmetic

machine epsilon:

## Properties of Floating Point Systems

All FP systems are characterized by several important numbers

- 1) a smallest normalized number (underflow)

$$\text{UFL} = 1.000\dots x \beta^{E_{\min}}$$

- 2) a largest normalized number (overflow)

$$\text{OFL} = d.\text{dddddd}\dots x \beta^{E_{\max}} \quad (\text{where } d = \beta - 1)$$

- 3) Zero:

$$0 = 0.000000\dots x \beta^0$$

- 4) Machine epsilon  $\epsilon_{\text{mach}}$

the smallest number such that  $1 + \epsilon_{\text{mach}} > 1$

- 5) Inf and NaN:

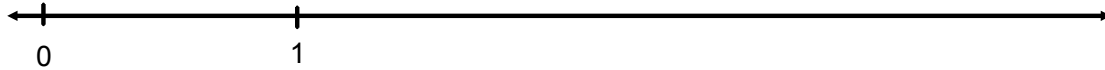
- 6) Subnormal numbers

$$f = 0.d_1 d_2 \dots d_p \times \beta^{E_{\min}}$$

## Binary Systems:

Consider the toy 2-digit precision base 2 system

$$f = \pm d_1.d_2 \times 2^E \quad \text{with } E \text{ in } [-2,2]$$



## Real systems: IEEE 754 Binary floating point systems

Single Precision: total storage 32 bits

Exponent 8 bits:  $E = [-126, 127]$

Fraction 23 bits ( $p=24$ )

S EEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFF  
01            89                            31

$$\text{OFL} = 2^{127} \sim 3.4 \times 10^{38}$$

$$\text{UFL} = 2^{-126} \sim 1.2 \times 10^{-38}$$

$$\epsilon_{\text{mach}} = 2^{-23} \sim 1.2 \times 10^{-7}$$

$$\text{smallest subnormal } 2^{-149} \sim 1.4 \times 10^{-45}$$

## Real systems: IEEE 754 Binary floating point systems

Double Precision: total storage 64 bits

Exponent 11 bits:  $E = [-1022, 1024]$

Fraction 52 bits ( $p=53$ )

```
S EEEEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
01           11 12                                           63
```

$$\text{OFL} = 2^{1024} \sim 1.8 \times 10^{308}$$

$$\text{UFL} = 2^{-1022} \sim 2.2 \times 10^{-308}$$

$$\epsilon_{\text{mach}} = 2^{-52} \sim 2.2 \times 10^{-16}$$

$$\text{smallest subnormal } 2^{-1074} \sim 4.9 \times 10^{-324}$$

## Real systems: IEEE 754 Binary floating point systems

Matlab access to IEEE numbers

`realmax`: returns OFL

`realmin`: returns UFL

`eps`: returns  $\epsilon_{\text{mach}}$

defaults are double precision: for single precision use

`realmax('single')`, `realmin('single')`, `eps('single')`

`eps` is actually a more general function

`eps(x)`: returns 1 ulp (next closest FP number  $> x$  which differs by 1 "unit in the last place")

e.g. `eps(realmin)` returns the smallest non-normal number

`errulps = abs(f-fhat)/eps(f)` returns the absolute error in ulps

**Big Points:**

Floating point math is not commutative or associative  
Floating point errors can destroy precision  
even IEEE double precision can break

Some Examples:

**Floating Point and Truncation Error**

In many numerical schemes both truncation error and floating point error can contribute to the overall error

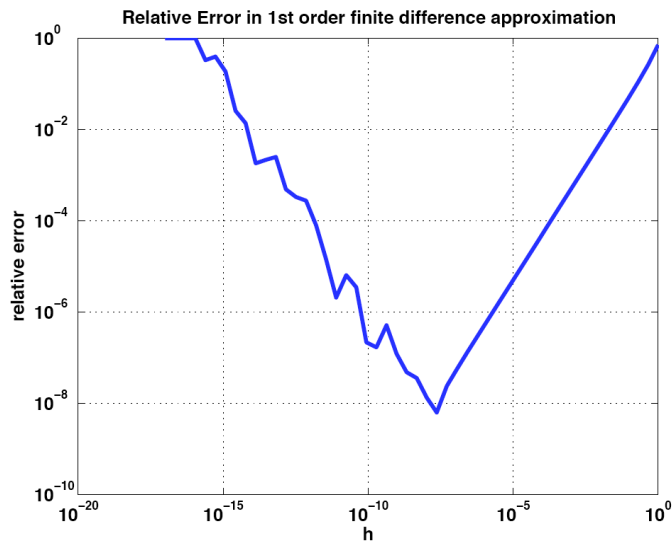
Example #1: Finite Difference approximation to  $f'(x)$

let  $f(x)=\exp(x)$ ,  $f(1)=e$ ,  $f'(1)=e$

## Floating Point and Truncation Error

Example #1: Finite Difference approximation to  $f'(x)$

let  $f(x)=\exp(x)$ ,  $f(1)=e$ ,  $f'(1)=e$



## Floating Point and Truncation Error

Example #2: The great  $\exp(x)$  challenge:

given the Taylor polynomial expansion of  $\exp(x)$  around 0

$\exp(x) \sim$

provide a matlab function `estimateExp(x)` such that the average relative error is less than 8 ulps over the range  $x = [-20,20]$  (that's not that hard...overachievers can do this on  $x=[-700,700]$ ).

A simple (but not ideal way to try)

```
N=100;
n=0:N;
estimateExp = @(x) sum(x.^n./factorial(n))
```