

## COMP3320/COMP6464: Floating Point Numbers

Alistair Rendell

---

COMP3320 Lecture 3-0 Copyright © 2008 The Australian National University

### 3.1 Rational

*It is extremely important that all scientific programmers have a basic understanding of numerical data representation and error propagation ... catastrophic failure of the Ariane 5 rocket in 1996 and the inability of Patriot missile systems to reach their targets during the 1991 Gulf war were both attributed to numerical computing errors*

- Ariane 5 Explosion (1996)
  - data conversion of too large number (trying to stuff a 64-bit number into a 16-bit field)
  - problem came to light as Ariane 5 was faster than Ariane 4.
- Patriot-Scud fails again (1991)
  - due to inaccurate calculation of the time since computer boot time
- Mars Orbiter loss (1999)
  - mixture of miles and meters
- See <http://www.zenger.informatik.tu-muenchen.de/persons/huckle/bugse.html>

---

COMP3320 Lecture 3-1 Copyright © 2008 The Australian National University

### 3.2 Floating Point Numbers

- Most scientific problems are concerned with continuous phenomena, e.g. time, distance, velocity, temperature
  - i.e. real numbers not integers
- Computers operate in an exact discrete world
- An understanding of how floating point arithmetic is handled on computers is fundamental to scientific computing
  - all computers (CPUs) of interest have separate floating point and integer arithmetic units
- Numerical analysis deals with the design and analysis of the behaviour of algorithms for solving scientific problems

---

COMP3320 Lecture 3-2 Copyright © 2008 The Australian National University

### 3.3 Sources of Error

- Before computation begins
  - modeling
  - empirical measurements
  - previous computations
  - mistakes
- During computation
  - approximation error (truncation or discretization error)
  - rounding error
- Accuracy of final result inevitably reflects combination of approximations, and perturbations may be amplified by nature of problem or algorithm

---

COMP3320 Lecture 3-3 Copyright © 2008 The Australian National University

### 3.4 Approximation Error

- Difference between true result (for actual input) and result that would be produced by given algorithm using exact arithmetic, e.g.

- approximation of a series

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots$$

$$e^x \approx \sum_{n=0}^N \frac{x^n}{n!} \quad (N \text{ is large})$$

- numerical derivative

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \quad (h \text{ is small})$$

### 3.5 Rounding Error

- Difference between result produced by given algorithm using exact arithmetic and result produced by same algorithm using rounded arithmetic

- due to inexactness in representation of real numbers and arithmetic operations upon them

- effects like  $1/3 + 1/3 + 1/3 = 0.9999$

### 3.6 Total Computation Error

- Total Computation Error = Approximation Error + Rounding Error

- but usually one of these is dominant (see labs)

### 3.7 Absolute and Relative Error

- Absolute Error

$$\begin{aligned} \text{Error Absolute} &= \text{Value Approximate} - \text{Value True} \\ \Rightarrow \text{Value Approximate} &= (\text{Value True})(1 + \text{Relative Error}) \\ V_a &= V_t(1 + \epsilon) \end{aligned}$$

- Relative Error

$$\begin{aligned} \text{Error Relative} &= (\text{Error Absolute})/(\text{Value True}) \\ &\approx (\text{Absolute Error})/(\text{Value Approximate}) \end{aligned}$$

- True value is usually unknown

- so we must estimate or bound error rather than compute it exactly

- Likewise relative error is often taken to be relative to approximate value, rather than true value

### 3.8 Floating Point Numbers

- Characterised by
  - $\beta$ : base or radix
  - $t$ : the precision (significant figures)
  - $[L, U]$ : exponent range

- Number (e.g.  $-1.23456 \times 10^{-13}$ ) represented as

$$x = \pm \left( d_0 + \frac{d_1}{\beta^1} + \frac{d_2}{\beta^2} + \frac{d_3}{\beta^3} + \dots + \frac{d_{t-1}}{\beta^{t-1}} \right) \beta^e$$

$$0 \leq d_i \leq \beta - 1$$

$$i = 0, 1, 2, \dots, t - 1$$

$$L \leq e \leq U$$

- $d_0 d_1 d_2 \dots d_{t-1}$  mantissa ( $d_1 d_2 \dots d_{t-1}$  behind decimal place)
- $e$  is the exponent

### 3.9 Typical Floating Point Systems

- Most computers use binary ( $\beta = 2$ ) arithmetic

System	$\beta$	$t$	$L$	$U$
IEEE SP	2	24	-126	127
IEEE DP	2	53	-1022	1023
Cray XMP	2	48	-16383	16384
HP Calculator	10	12	-499	499
IBM Mainframe	16	6	-64	63

- IEEE by far most common
- Note trade-off between  $t$  and  $[L, U]$

### 3.10 Normalization

- Floating point system is *normalized* if leading digit  $d_0$  is always nonzero
  - mantissa  $m$  of nonzero floating point number always

$$1 \leq m < \beta$$

- Why
  - unique representation of each number
  - no digits wasted on leading zero
  - in binary system the leading bit need not be stored

### 3.11 Properties of Floating Point Systems

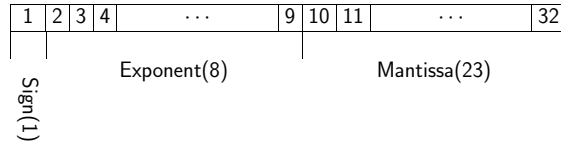
- FP number system is finite and discrete
- Number of normalized FP numbers

$$2(\beta - 1)\beta^{t-1}(U - L + 1) + 1$$

- Smallest and largest positive normalized number
  - underflow level = UFL =  $\beta^L$
  - overflow level = OFL =  $\beta^{U+1}(1 - \beta^{-t})$
- FP numbers are equally spaced only between successive powers of  $\beta$
- Not all real numbers are exactly representable
  - those that are, are called *machine numbers*

### 3.12 IEEE Single Precision (32 Bits)

- $\beta = 2$ ,  $t = 24$  (normalized),  $L = -126$ ,  $U = +127$   
 $\Rightarrow 2(2)23(127+126+1)+1$  numbers



#### Special Cases

- zero
- infinity (plus and minus)
- not a number (nan)
- sub-normal numbers

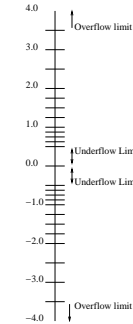
### 3.13 Special Cases: Single Precision

0	0000 1110	1010 0011	1111 1011	0100 111
Valid IEEE Single Precision Number				
0	0000 0000	0000 0000	0000 0000	0000 000
IEEE +0				
1	0000 0000	0000 0000	0000 0000	0000 000
IEEE -0				
0	1111 1111	0000 0000	0000 0000	0000 000
IEEE +∞				
1	1111 1111	0000 0000	0000 0000	0000 000
IEEE -∞				
	1111 1111	NON ZERO		
IEEE Not A Number (NaN)				

### 3.14 Example Floating Point System

- $\beta = 2$ ,  $t = 3$ (normalized),  $L = -1$ ,  $U = +1$  (25 numbers)

Sign	$d_0$	$d_1$	$d_2$	$e$	decimal
0	1	1	1	1	3.5
0	1	1	0	1	3.0
0	1	0	1	1	2.5
0	1	0	0	1	2.0
0	1	1	1	0	1.75
0	1	1	0	0	1.50
0	1	0	1	0	1.25
0	1	0	0	0	1.0
0	1	1	1	-1	0.875
0	1	1	0	-1	0.75
0	1	0	1	-1	0.625
0	1	0	0	-1	0.5
0	0	0	0	0	0.0



- OFL =  $(1.11)_2 \times 2^1 = 3.5_{10}$ , UFL =  $(1.00)_2 \times 2^{-1} = 0.5_{10}$
- Note gray and unequally spaced

### 3.15 Rounding Rules

- Non-machine numbers must be approximated to "nearby" FP number
  - termed *rounding* and introduces *rounding error*
  - denote as  $fl(x)$
- Rounding rules
  - *chop*: truncate  $x$  after  $(t - 1)$  digit (round toward zero or round down)
  - *round to nearest*
- IEEE uses rounding to nearest

### 3.16 Machine Precision

- Accuracy of FP system characterized by quantity called "unit round-off error" or "machine precision", or "machine epsilon", denoted by  $\epsilon_{\text{mach}}$

- chopping:  $\epsilon_{\text{mach}} = \beta^{1-t}$
- nearest:  $\epsilon_{\text{mach}} = 1/2\beta^{1-t}$

- Maximum relative error in representing real number  $x$  is given by:

$$\left| \frac{fl(x) - x}{x} \right| \leq \epsilon_{\text{mach}}$$

### 3.18 Machine Precision: Toy System

- Round to nearest:  $\epsilon_{\text{mach}} = 1/2\beta^{1-t} = 0.125$

$$Err1 = \left| \frac{fl(x) - x}{x} \right| \quad Err2 = \left| \frac{fl(x) - x}{fl(x)} \right|$$

Mantissa	Exponent	$x$	$fl(x)$	$Err1$	$Err2$		
1 0 0	1.000	-1	0.500	0.562	0.111	0.125	
1 0 1	1.250	-1	0.500	0.625	0.688	0.091	0.100
1 1 0	1.500	-1	0.500	0.750	0.812	0.077	0.083
1 1 1	1.750	-1	0.500	0.875	0.938	0.067	0.071
1 0 0	1.000	0	1.000	1.000	1.125	0.111	0.125
1 0 1	1.250	0	1.000	1.250	1.375	0.091	0.100
1 1 0	1.500	0	1.000	1.500	1.625	0.077	0.083
1 1 1	1.750	0	1.000	1.750	1.875	0.067	0.071
1 0 0	1.000	1	2.000	2.000	2.250	0.111	0.125
1 0 1	1.250	1	2.000	2.500	2.750	0.091	0.100
1 1 0	1.500	1	2.000	3.000	3.250	0.077	0.083
1 1 1	1.750	1	2.000	3.500			

### 3.17 Machine Precision

- For IEEE FP systems

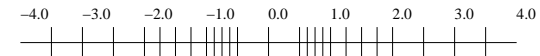
- Single Precision:  $\epsilon_{\text{mach}} = 2^{-24} \approx 10^{-7}$
- Double Precision:  $\epsilon_{\text{mach}} = 2^{-53} \approx 10^{-16}$

- IEEE SP and DP have  $\approx 7$  and 16 decimal digits of precision respectively
- Machine precision and UFL are VERY different

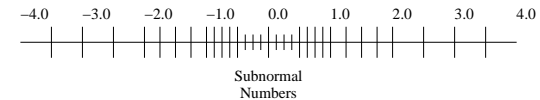
$$0 < \text{UFL} < \epsilon_{\text{mach}} < \text{OFL}$$

### 3.19 Subnormal Numbers and Gradual Underflow

- Normalization causes gap around zero



- Subnormal or denormalized numbers permit leading digit to be zero only for minimum exponent.



- Subnormal numbers extend range of magnitude representable, but they have less precision and unit roundoff is the same
- Allows system to exhibit gradual underflow

### 3.20 Errors and Floating Point Arithmetic

- Require,  $x \text{ flop } y = fl(x \text{ flop } y)$ 
  - IEEE systems achieve this as long as  $(x \text{ flop } y)$  is within the range of the FP system
- Some familiar laws of real arithmetic are not valid!
  - FP addition and multiplication are commutative ( $a \times b = b \times a$ ), but NOT associative ( $(a + b) + c \neq a + (b + c)$ )
  - e.g. if  $\delta$  is a small positive number slightly less than  $\epsilon_{\text{mach}}$ 
$$(1 + \delta) + \delta = 1 + \delta = 1 \text{ BUT } 1 + (\delta + \delta) > 1$$

### 3.21 Cancellation

- Subtraction of two  $t$ -digit numbers with same sign and similar magnitude yields fewer than  $t$  digits
  - leading digits of two numbers cancel, e.g.
$$\begin{array}{r} 192.403 \\ -192.275 \\ \hline 000.128 \end{array} = 1.28000 \times 10^{-1}$$
- Exact result but cancellation often implies serious loss of information
  - e.g.  $\delta < \epsilon_{\text{mach}}$ 
$$(1 + \delta) - (1 - \delta) = 1 - 1 = 0$$

correct result of  $2\delta$  entirely lost

### 3.22 Error Propagation: Addition/Subtraction

- Addition/subtraction
  - add or subtract absolute errors
$$\begin{array}{ll} b = 2.0 \pm 0.1 & \epsilon_b = 0.1/2.0 = 0.05 \\ c = 3.0 \pm 0.3 & \epsilon_c = 0.3/3.0 = 0.10 \\ a = b + c = 5.0 \pm 0.4 & \epsilon_a = 0.4/5.0 = 0.08 \end{array}$$

relative error averaged

$$\epsilon_a = \frac{b\epsilon_b + c\epsilon_c}{b + c}$$

### 3.23 Error Propagation: Multiplication/Division

- Multiplication/Division
  - add or subtract relative errors
$$\begin{array}{ll} b = 2.0 \pm 0.1 & \epsilon_b = 0.1/2.0 = 0.05 \\ c = 3.0 \pm 0.3 & \epsilon_c = 0.3/3.0 = 0.10 \\ a = b \times c & \\ = (2.0 \pm 0.1) \times (3.0 \pm 0.3) & \\ = 6.0 \pm (0.6 + 0.3) & \epsilon_a = 0.9/6.0 = 0.15 \\ = 6.0 \pm 0.9 & \end{array}$$

relative error summed

### 3.24 Real Life Error Propagation

- Good algorithms usually give

$$\begin{aligned}\Sigma \text{ errors} &= \sqrt{N}\epsilon \\ N &= \text{Number of operations} \\ \epsilon &= \text{Error per operation}\end{aligned}$$

- 1GHz UltraSparc performs  $2 \times 10^9$  floating point operations per second
  - Compute for 1 hour  $\Rightarrow N = 7.2 \times 10^{12}$  operations
  - $\sqrt{N} \approx 3 \times 10^6$
  - compare to single precision  $\epsilon_{\text{mach}} \approx 10^{-7}$

*Virtually all accuracy is lost*

- Need to be very careful about error propagation

### 3.25 Assessing Rounding Errors

- Examine effect of using higher precision
- Consider famous example from Rump in 1988 (IBM S/370)

$$f = 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + a/(2b)$$
$$a = 77617.0, b = 33096.0$$

single precision  $f = 1.172603\dots$   
double precision  $f = 1.1726039400531\dots$   
extended precision  $f = 1.172603940053178\dots$

However true result is

$$f = -.82739605994682135 \pm 5 \times 10^{-17}$$

Even the sign is wrong!

- While this is an extreme example of catastrophic cancellation in an inherently unstable function

*there must be a better way!*

### 3.26 Interval Arithmetic: A New Floating Point Paradigm

- We are all familiar with representation of uncertainty using the  $(x \pm \epsilon)$  notation, but this is cumbersome for computation, instead we define an interval as
  - a range of numbers bounded by the intervals endpoints
  - e.g. the range 378 to 432 would be written as [378,432].
- Evaluating a function in an interval, gives another interval
  - how long to double your money if invested at a compound interest rate of between [3,6]% per annum
  - answer is the interval [11yr. 7mo., 23yr.2mo.]
- Interval computations do not always increase interval width
  - evaluate  $f(x) = \ln(x)/x$  over interval  $X=[2.716,2.718]$
  - answer is the interval [0.3678793,0.3678795]

### 3.27 Improved Numerical Accuracy

- Two common ways to improve numerical accuracy and reduce interval width
  - reduce interval width of input values
  - increase the number of interval data values used to compute interval bounds
- First is obvious the second is more subtle
  - given n interval observations of the same true value, the best way to compute a narrow interval bound is not to compute the average, but rather the intersection
  - every observation must contain the correct value, so must their intersection

### 3.28 Support for Interval Arithmetic

- C++ and Fortran95 contain language constructs to enable abstract mathematical data types
  - use of classes/modules, user defined types and operator overloading
- This is okay for matrices, but less good for intervals
  - Sun's Fortran95 has been augmented to include an "interval" data type
  - there is a proposal to include intervals in future Fortran standards
  - UltraSPARC III includes some hardware support for interval instructions

### 3.29 Use of Intervals

The use of intervals in numerical computation is still very much an area of research, although there is a growing argument that projects like ASCI should require intervals (see Gustafson).

- Use to track rounding errors
  - assign to each variable an interval related to machine precision
  - propagate computation
- Use to accurately reflect uncertainty
  - used in refining the value of the gravitational constant G
- Fundamentally new approaches
  - solution of non-linear equations
  - global optimisation

We may briefly explore the use of intervals in the labs