# Numerical Methods

## Lecture 3

CS 357 Spring 2014

David Semeraro

# Taylor's theorem for f(x)

If the function $f$ possesses continuous derivatives of orders 0,1,2,…,(n+1) in a closed interval $I = [a, b]$, then for any c and x in $I$,

$$f(x) = \sum_{k=0}^{n} \frac{f^{k}(c)}{k!}(x - c)^{k} + E_{n+1}$$

Where the error term can be given in the form

$$E_{n+1} = \frac{f^{(n+1)}(\varepsilon)}{(n+1)!}(x - c)^{n+1}$$

# Alternating Series (from last week)

If $a_1 \geq a_2 \geq a_3 \geq \cdots \geq a_n \geq \cdots \geq 0$ for all $n$ and $\lim_{n\to\infty} a_n = 0$, then the alternating series:

$$a_1 - a_2 + a_3 - a_4 + \cdots$$

Converges; that is,

$$\sum_{k=1}^{\infty}(-1)^{k-1}a_k = \lim_{n\to\infty}\sum_{k=1}^{n}(-1)^{k-1}\,a_k = \lim_{n\to\infty} S_n = S$$

Where $S$ is its sum and $S_n$ is the $n^{th}$ partial sum. Moreover, for all $n$.

$$|S - S_n| \leq a_{n+1}$$

# Alternating Series (from last week)

Partial Sums $S_n$

$$S_1 = a_1$$
$$S_2 = a_1 - a_2$$
$$S_3 = a_1 - a_2 + a_3$$
$$S_4 = a_1 - a_2 + a_3 - a_4$$
$$\dots$$

# Alternating Series Example

- How many terms are needed to approximate $\sin 1$with an error less than $\frac{1}{2}\times 10^{-6}$?

Expand about c = 0.

$$\sin x \approx \sin(0) + \cos(0)\,x - \sin(0)\frac{x^2}{2!} - \cos(0)\,\frac{x^3}{3!} + \cdots$$

Since $\sin(0) = 0$ and $\cos(0) = 1$:

$$\sin(x) = 1 - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

# Alternating Series Example

$$\sin(1) = 1 - \frac{1}{3!} + \frac{1}{5!} - \frac{1}{7!} + \cdots$$

By the alternating series theorem, the error does not exceed the first neglected term:

$$|S - S_n| \leq a_{n+1}$$

After n terms the first neglected term is $\frac{1}{(2n+1)!}$.

$$\frac{1}{(2n+1)!} < \frac{1}{2} \times 10^{-6}$$

# Alternating Series Example

$$\frac{1}{(2n+1)!} < \frac{1}{2} \times 10^{-6}$$

Using base 10 logarithms…

$$\log[(2n+1)!] > \log 2 + 6 = 6.3$$

$$\log 10! \approx 6.6 \;\; \rightarrow n \geq 5$$

Now for something different…

Floating Point Representation

# Floating Point

37.96389

Integer Part

Fractional Part

## Normalized Scientific Notation

$$37.96389 = 0,3796389 \times 10^2$$

*Not* zero

Fraction $\times\ 10^n$

# Floating Point (base 10)

$$(a_n a_{n-1} \dots a_1 a_0 . b_1 b_2 \dots)_{10} = \sum_{k=0}^{n} a_k 10^k + \sum_{k=1}^{\infty} b_k 10^{-k}$$

- Some numbers have infinite number of digits in the fractional part.
  - $\pi = 3.14159 \dots$
  - $\sqrt{2} = 1.41421 \dots$

# Other Bases ($\beta$)

- Binary ($\beta = 2$)
- Octal ($\beta = 8$)
- Hex ($\beta = 16$)

$$(a_n a_{n-1} \ldots a_1 a_0 . b_1 b_2 \ldots)_\beta = \sum_{k=0}^{n} a_k \beta^k + \sum_{k=1}^{\infty} b_k \beta^{-k}$$

(See Appendix B in Text)

# Integer Conversion

- Decimal to Binary (example)

$$(N)_{10} = a_j 2^j + \cdots + a_1 2 + a_0$$

- Compute $\frac{(N)_{10}}{2} = Q + R$
  - $Q = a_j 2^{j-1} + \cdots + a_2 2 + a_1$
  - $R = a_0$ ( $a_0$ is the *remainder* in long division)
- Repeat process on successive $Q$ to obtain remaining digits.

# Integer Conversion

- Convert 743 to binary.
- $\frac{743}{2} = 371$ remainder 1.
- $Q = 371; R = 1 = a_0$
- $a_0 = 1$
- Repeat

$$(743)_{10} = (1011100111)_2$$

| j | Q | $(a_j)$ |
|---|---|---|
| 0 | 371 | 1 |
| 1 | 185 | 1 |
| 2 | 92 | 1 |
| 3 | 46 | 0 |
| 4 | 23 | 0 |
| 5 | 11 | 1 |
| 6 | 5 | 1 |
| 7 | 2 | 1 |
| 8 | 1 | 0 |
| 9 | 0 | 1 |

# Fractional Part

$$x = \sum_{k=1}^{\infty} c_k \beta^{-k} = (0.c_1 c_2 \ldots)_\beta$$

$$\beta x = (c_1.c_2\, c_3 \ldots)_\beta$$

Equate the integer parts of both sides of the = sign.

$$c_1 = I(\beta x)$$

Where $I(y)$ is the integer part of y.

# Fractional Conversion

$$d_0 = x$$

$$d_1 = F(\beta d_0) \qquad c_1 = I(\beta d_0)$$

$$d_2 = F(\beta d_1) \qquad c_2 = I(\beta d_1)$$

$$\ldots \qquad \ldots$$

- Start with x
- Multiply x by $\beta$
- Set coefficient to integer part of product.
- Repeat with fractional part of product

# Fractional Conversion

- Convert 0.100 to binary.

$$2 \times 0.100 = 0.200 \rightarrow c_1 = 0$$
$$2 \times 0.200 = 0.400 \rightarrow c_2 = 0$$
$$2 \times 0.400 = 0.800 \rightarrow c_3 = 0$$
$$2 \times 0.800 = 1.600 \rightarrow c_4 = 1$$
$$2 \times 0.600 = 1.200 \rightarrow c_5 = 1$$
$$2 \times 0.200 = 0.400 \rightarrow c_6 = 0$$
$$(\mathbf{0.1})_{\mathbf{10}} = (\mathbf{0.0001100110011\ldots})_{\mathbf{2}}$$

# Normalized Floating Point

- Real decimal number x can be written:

$$x = \mp 0.d_1 d_2 d_3 \ldots \times 10^n$$

- Or

$$x = \mp r \times 10^n \ \left(\tfrac{1}{10} \le r < 1\right)$$

- $r$ – normalized mantissa in $[\frac{1}{10}, 1)$
- $n$ – exponent
- $d_1 \neq 0$

# Normalized Floating Point

- Real binary number x can be written:

$$x = \mp 0.b_1 b_2 b_3 \ldots \times 2^n$$

- Or

$$x = \mp q \times 2^m \ \left(\tfrac{1}{2} \leq q < 1\right)$$

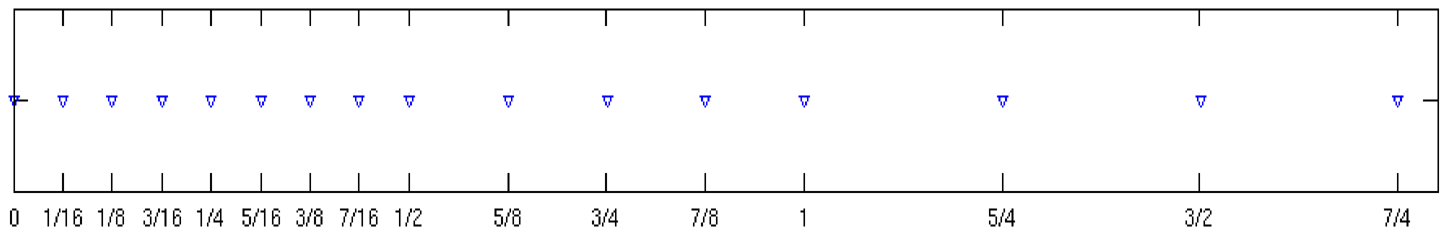
- q— normalized mantissa in $\left[\frac{1}{2}, 1\right)$
- $n$ − exponent
- $b_1 \neq 0$

# Computer Representation

- Finite Word Length (number of bits per word)
  - Finite number of digits per number
  - Irrational numbers can not be represented
  - Numbers may be too big or too small
- 1 word per number in single precision
- 2 or more words per number in extended precision.

# Computer Representation

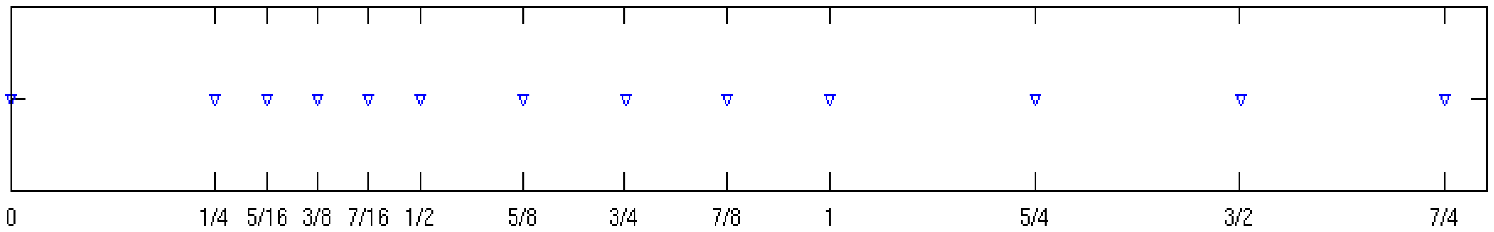
- *Machine numbers* are a discrete set.
- Consider $x = \mp(0.b_1 b_2 b_3) \times 2^{\mp m}$




- $x = \mp q \times 2^m; -1 \leq m \leq 1$
- $m$ outside permissible range – *overflow or underflow.*

# Computer Representation

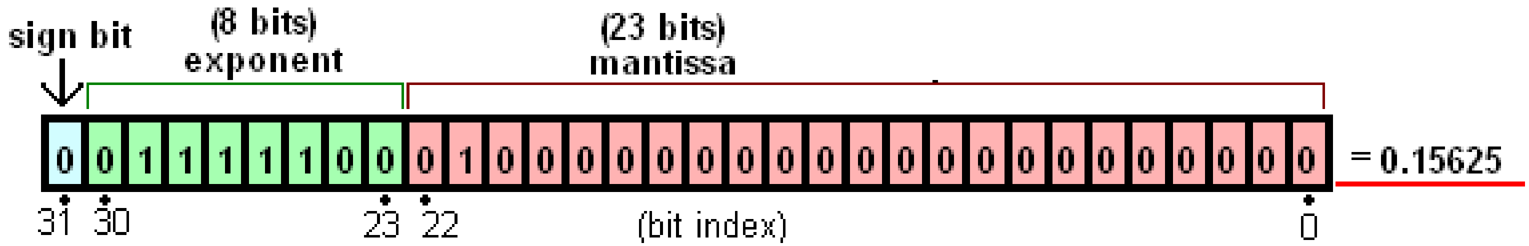
- Numbers $< \frac{1}{16}$ underflow to zero.
- Numbers $> \frac{7}{4}$ overflow to machine infinity.
- Allowing only normalized numbers $(b_1 = 1)$ creates *hole at zero.* 1/8, 1/16, and 3/16 are lost.



# Computer Representation

- Single-precision
  - 32 bit word
  - Most reals cannot be expressed as floating point number due to infinite decimal or binary representation.
- Splitting up the bits $\mp q \times 2^m$
  - Sign of $q$ – 1 bit
  - Integer $|m|$ - 8 bits (sign contained in the 8 bits.)
  - Number $q$ – 23 bits

# Single Precision




Standard single precision floating point

$$(-1)^s \times 2^{c-127} \times (1.f)_2$$

- Bit 31 contains s, sign of mantissa.
- Bits 23-30 contain c in $2^{c-127}$
- Bits 0-22 contains $f$ from $(1.f)_2$

# Representation of the Mantissa

- The first bit in the mantissa of a normalized floating point number is always 1.
- In the 1-plus form $(1.f)_2$ the right most 23 bits of the word represent $f$ and the 1 is hidden.
- So you get 24 bits of accuracy while using 23.
- $1 \leq (1.f)_2 \leq 2 - 2^{-23}$

# Exponent

- $0 < c < 255$ ( 0 and 255 reserved )
- $-126 \ \leq c - 127 \ \leq 127$ (actual exponent)

# Representation

- Largest single precision floating point number representable is
$$(2-2^{-23})2^{127} \approx 2^{128} \approx 3.4 \times 10^{38}$$
- Smallest  positive number representable is
$$2^{-126} \approx 1.2 \times 10^{-38}$$
- Machine epsilon $\epsilon = 2^{-23} \approx 1.2 \times 10^{-7}$ is the smallest number such that $1 + \epsilon \neq 1$.

# Representation of real X

- If x is zero use full word of zero bits. (possible sign bit)
- For nonzero x
  - Assign sign bit
  - Convert integer and fractional part of |x| to binary
  - 1-plus normalize the result by shifting binary point so that first bit to left of point is 1. All bits to left of this 1 are zero.
  - Adjust the exponent to reflect the bit shift in the mantissa.
  - Determine C from the current exponent.

# Example from book

- Determine single precision machine representation of -52.234375
- Negative number so sign bit is 1.
- Convert 52 to binary (see integer conversion example)
  - $(52.)_{10} = (110100.)_2$
- Convert 0.234375 to binary (see fractional conversion example)
  - $(.234375)_{10} = (.001111)_2$

# Example from book

- $(52.234375)_{10} = (1.101000011110)_2 \times 2^5$
  - After one-plus normalization
  - $(.101000011110)_2$ is the stored mantissa
- Exponent is $(5)_{10}$
  - We have $c - 127 = 5 \rightarrow c = 132$
  - Stored exponent: $(132)_{10} = (10000100)_2$
- $[1\color{blue}{10000100}\color{green}{10100001111000000000000}]_2$

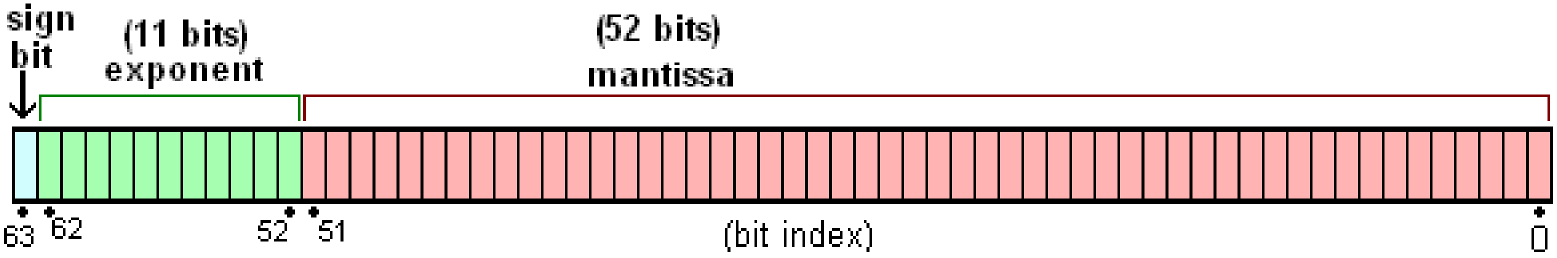
# Going the other way

- $[01000101110111110010000000000000]_2$
- Exponent $c = (10001011)_2 = (139)_{10}$
  - Exponent is $c - 127 = 12$
- Mantissa in one-plus form is
$$(1.101111001)_2$$
- Combining exponent and mantissa
$$(1.101111001)_2 \times 2^{12} = (1101111001000)_2$$
$$= (15710)_8$$
$$= 0 \times 1 + 1 \times 8 + 7 \times 8^2 + 5 \times 8^3 + 1 \times 8^4$$
$$= 7112$$

# Special Cases

- denormalized/subnormal numbers: use 1 extra bit in the mantissa
  - exponent is now -126 (less precision, more range), indicated by $00000000_2$ in the exponent field
- two zeros: +0 and -0 (0 mantissa, 0 exponent)
- two ∞'s: +∞ and -∞
- ∞ (0 mantissa, $11111111_2$ exponenet)
- NaN (any mantissa, $11111111_2$ exponent)
- see appendix C.1 in NMC 7th ed.

# Double precision




- 1-bit sign
- 11-bit exponent
- 52-bit mantissa
- single-precision: about 6 decimal digits of precision
- double-precision: about 15 decimal digits of precision
- m = c - 1023

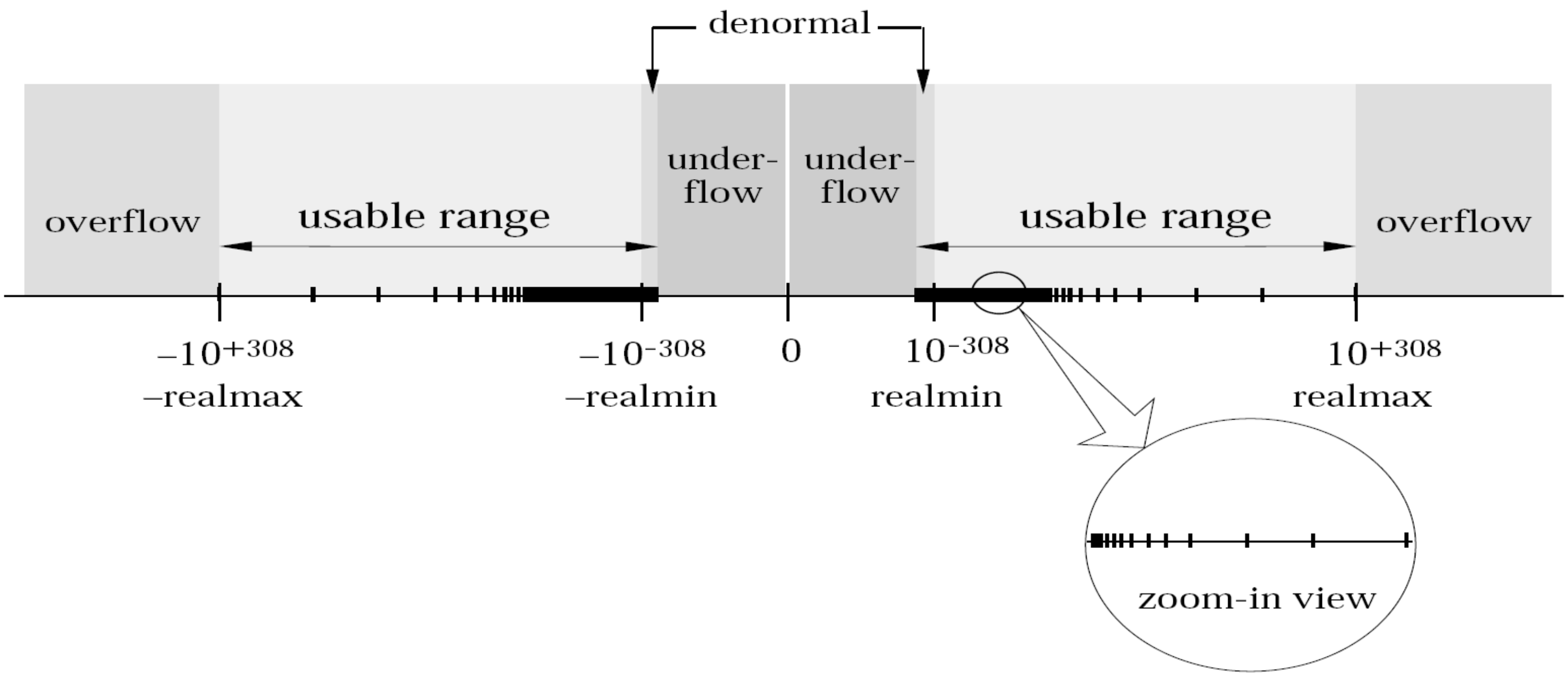
# Range

| Type | Range | Approx. Range |
|---|---|---|
| Single | $-3.4 \times 10^{38} \leq x \leq -1.18 \times 10^{-38}$ | $2^{-126} \rightarrow 2^{128}$ |
| | $0$ | |
| | $1.18 \times 10^{-38} \leq x \leq 3.4 \times 10^{38}$ | |
| Double | $-1.8 \times 10^{308} \leq x \leq -2.23 \times 10^{-308}$ | $2^{-1022} \rightarrow 2^{1024}$ |
| | $0$ | |
| | $2.23 \times 10^{-308} \leq x \leq 1.8 \times 10^{308}$ | |

```
>>> sys.float_info.max
1.7976931348623157e+308
>>> sys.float_info.min
2.2250738585072014e-308
>>>
```

# Number Line

## Floating Point Number Line



# Computer Representation

- *Roundoff* occurs when digits in a decimal point (0.3333...) are lost (0.3333) due to a limit on the memory available for storing one numerical value.
- *Truncation* error occurs when discrete values are used to approximate a mathematical expression.

# Uncertainty

Errors in input data can cause uncertain results

- Input data can be experimental or rounded. leads to a certain variation in the results
- ***Well-conditioned***: numerical results are insensitive to small variations in the input
- ***Ill-conditioned***: small variations lead to drastically different numerical calculations (a.k.a. poorly conditioned)
- L.

# Uncertainty

Need to…

1. Solve a problem so that the calculation is not susceptible to large roundoff error
2. Solve a problem so that the approximation has a tolerable truncation error

How?

- Incorporate roundoff-truncation knowledge into
  - The mathematical model
  - The method
  - The algorithm
  - Software design
- Utilize awareness of uncertainty to interpret results.