A.A. 2010/2011

Calcolo Scientifico per l'Informatica
- Laboratory Class -
1

Davide Fugazza
fugazza@elet.polimi.it

Outline

Outline

MATLAB®: a quick review

A list of some basics:

- MATLAB® user interface;
- variable definition: scalars, vectors and matrices, chars;
- vector and data matrix manipulation;
- code modularity: functions and scripts;
- data plots and simple graphics commands;
- I/O: loading and saving files, variables, and data.

These Labs are not intended to provide an introduction to MATLAB®. The student is supposed to have already developed a basic knowledge of the environment throughout his/her academic carrier.

Outline

Introductory examples

Cleve Moler's code sequence (From [4])

```
>>format long
>> a = 4/3
a =
   1.333333333333333

>> b = a - 1
b =
   0.333333333333333

>> c = 3*b
c =
   1.000000000000000

>> e = 1 - c
e =
    2.220446049250313e-016
```

Introductory examples, cont'd

Compare the results of the following code segments:

Example 1

```
>> x = 0;
>> while x ~= 1
>> x = x + 1/16;
>> [x, sqrt(x)]
>> end
```

Example 2

```
>> x = 0;
>> while x ~= 1
>> x = x + 0.1;
>> [x, sqrt(x)]
>> end
```

Is everything as expected?
We are dealing with floating point arithmetic!

Floating-Point Arithmetic

Technical computing environments use floating-point arithmetic, which involves a *finite set of numbers* with *finite precision*. The typical implementation represents the number in the *normalized form*:

$$\overline{x} = (-1)^{sign} \times significand \times base^{exponent}$$

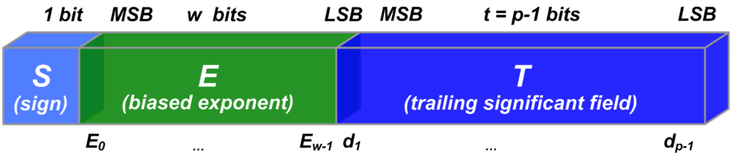
that can be equivalently written as:

$$\overline{x} = (-1)^{s} \times (1+f) \times b^{e}$$

where

- s is the sign: 0 $(+)$ or 1 $(-)$;
- b is the *base* of the number system ($e.g.$ for *hex* format $b = 16$);
- e is the *exponent*, $emin \leq e \leq emax$;
- $(1+f)$ is the *mantissa* or *fractional part*.

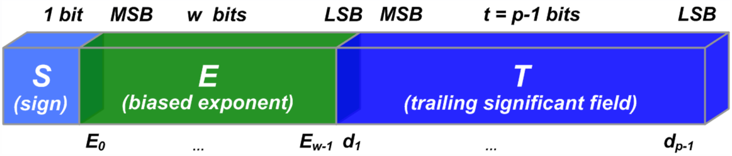
IEEE Std 754: Binary encodings



The overall representation is encoded in k bits

- 1 bit for the sign S
- w-bits for the biased exponent $E = e + bias$
- t-bits trailing significand field digit string $T = d_1 d_2 d_3 ... d_{p-1}$, where the leading bit (d_0) is implicitly encoded in the biased exponent E.

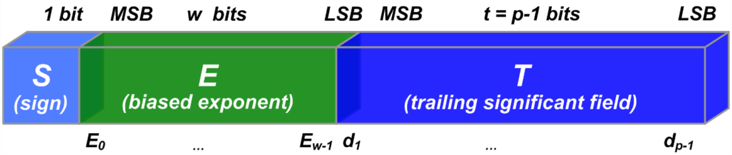
IEEE Std 754: Binary encodings



Exponent E: stored as unsigned number

- $L \leq e \leq U$, where $U = 2^{w-1} - 1$ and $L = 2 - 2^{w-1}$.
- $E = e + bias$, where the $bias$ is equal to $2^{w-1} - 1$.
- $E = 0$ is reserved to encode ∓ 0 $(T = 0)$ and subnormal numbers $(T \neq 0)$.
- $E = 2^w - 1$ is reserved to encode $\mp\infty$ $(T = 0)$ and $NaNs$ $(T \neq 0)$.
- w is a limitation on the achievable $range$ of the arithmetic.

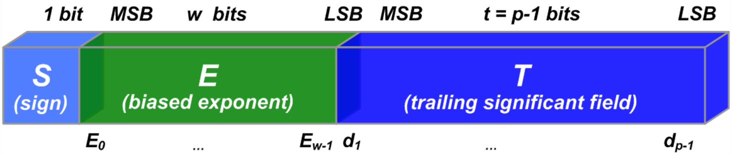
IEEE Std 754: Binary encodings



Exponent E: normal and subnormal numbers

- If $1 \leq E \leq 2^{w-1} - 2$, then $x = (-1)^S \times 2^{E-bias} \times (1.F)$, where $1.F$ is intended to represent the binary number created by prefixing F with and implicit leading 1 and binary point.
- If $E = 0$, then $x = (-1)^S \times 2^L \times (0.F)$. These numbers are called non-normalized or subnormal numbers.

IEEE Std 754: Binary encodings



Trailing significand field *T* (*Mantissa*)

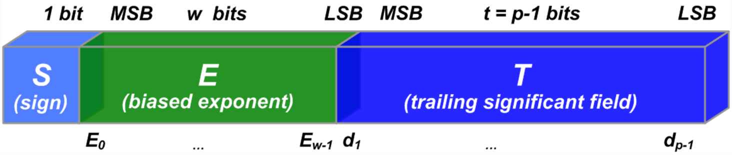
T is a string of bits in the form:

$$\frac{d_1}{2^1} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_{p-1}}{2^t}.$$

Note that:

- note that d_0 remains implicit and that d_1 is always $\neq 0$.
- p represents a limitation on the *precision* of the arithmetic.

IEEE Std 754: Binary encodings



Parameter	half	single	double	quad	binary(k)
k, storage width in bits	16	32	64	128	$1+w+t$
p, precision in bits	11	24	53	113	
exponent bias in bits	15	127	1023	16383	$2^{k-p-1}-1$
Sign bit	1	1	1	1	1
w, exp. field width in bits	5	8	11	15	
t, trail. sign. field width in bits	10	23	52	112	$k-w-1$

From IEEE Std 754™ - 2008 documentation.

IEEE Std 754: examples

for single precision (k = 32bits)

$$0\ 10000000\ 00000000000000000000000 = +1 \cdot 2^{128-127} \cdot 1.0_2 = 2$$
$$0\ 10000010\ 10001000000000000000000 = +1 \cdot 2^{130-127} \cdot 1.10001_2 = 12.25$$

$$0\ 00000000\ 00000000000000000000000 = 0$$
$$0\ 11111111\ 00000000000000000000000 = +Infinity$$
$$1\ 11111111\ 00000000000000000000000 = -Infinity$$
$$0\ 11111111\ 00000000000100000010000 = NaN$$

$$0\ 00000001\ 00000000000000000000000 = +1 \cdot 2^{1-127} \cdot 1.0_2 = 2^{-126}$$
$$0\ 00000000\ 10000000000000000000000 = +1 \cdot 2^{-126} \cdot 0.1_2 = 2^{-127}$$
$$0\ 00000000\ 00000000000000000000001 = +1 \cdot 2^{-126} \cdot 0.00\cdots01_2 = 2^{-149}$$

IEEE Std 754: hex format

In MATLAB every computation is performed in *double precision*.

- Use `format` and the appropriate options to set the output format (Note: this does not affect the way the computations are performed).

```
>> format long
>> x = 1

x =
     1
```

```
>> format hex
>> x = 1

x =
     3ff0000000000000
```

- Use `single` (`double`) commands to convert a number to single (double) precision.
- Use `num2hex` (`hex2num`) ... for conversion in the representations.

Refer to the related helps (`help format`,...) for further details!

IEEE Std 754: examples

The minimum and the maximum normalized machine's number can be obtained as follows:

```
>> minimum_real = realmin
```

```
>> maximum_real = realmax
```

- Note that `realmin` is different from eps.
- Any computation trying to produce a value larger than `realmax`, it is said to overflow. The result is called `Inf`.
- Any computation trying to produce a value smaller than `realmin`, it is said to underflow.
- For any computation trying to produce a value that is undefined the result is called `NaN` (*e.g.* `0/0`, `Inf/Inf`, `Inf-Inf`).

For double precision: realmin $= 2^{-1022}$, realmax $= (2 - eps) * 2^{1023}$, eps $= 2^{-52}$.

Outline

Machine epsilon: Exercise

Exercise 1 (From [2])

Evaluate the *epsilon machine* in the following three ways:

1. Implementing an *ad hoc* routine;
2. Executing the following code sequence (by Cleve Moler):

```
>> a = 4/3
>> b = a - 1
>> c = b + b + b
>> e = 1 - c
```

3. Using eps command.

Comment and justify the obtained results.

Floating-Points: Comments

Note that floating-point numbers are rational. The base determines the fractions that can be represented. For instance $1/10$ or $1/5$ can be exactly represented only in the decimal format.

```
>> x = 0.1
```

The mathematical value stored in `x` is not exactly `0.1`, because the fraction `1/10` in binary would require an infinite series!!

$$\frac{1}{10} = \frac{1}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{0}{2^{10}} + \frac{0}{2^{11}} + \cdots$$

Let's verify:

```
>> b = num2bin(quantizer('double'),0.1)
b =
1011111111111001100110011001100110011001100110011001100110011010
```

Floating-Points: Exercise

Exercise 2

Consider a binary floating-point arithmetic with $k = 6bits$ ($w = 3bits$ and $t = 2bits$), with rounding.

1. How many numbers can be represented in this arithmetic? What about the error due to binary representation?
2. Represent the following numbers: $2, 1.6, -1, -14.01$. Check the values with MATLAB.
3. Compare the results with the *double precision* format ($64bits$).
4. Which are the correspondent representations in hexadecimal format?
5. Convert the obtained results to *single precision*.

See also `help quantizer` and `help num2bin`. Comment the results.

Outline

Numerical Cancellation

Subtracting two nearly equal numbers, we loose a great deal of accuracy!

Example

```
>> a = 9.901020304e-9;
>> b = 9.0990055e-9;
>> c = 1.8500185e8;
>> s1 = (a+b) + c;
>> s2 = a + (b+c);
>> e = abs(s1-s2)
e =
    2.98023223876953le-008
```

Note: as a consequence of the loss of significant digits, the associative property is no more longer valid within floating-point arithmetic: *The order in which the operations are performed sometimes matters!*

Loss-of-Significance: Exercise

Exercise 3 (From [5])

Consider the following function:

$$f(x) = \frac{e^x - 1}{x}$$

1. Use De L'Hôpital's and Taylor's theorems to get an estimation of $f(x)$ around $x = 0$.
2. Experimentally evaluate $f(x)$ for values of x near zero (try with $2^{-k}, k \in [1, 30]$): what do you obtain for single and double precision? Explain the results.
3. Propose an approach to fix the problem (Hint: Use Taylor's expansion).
4. How many terms in a Taylor's expansion are needed to get single precision accuracy (7 decimal digits) for all $x \in [0, 1/2]$? How many terms are needed for double precision accuracy (14 decimal digits) over the same range?

Outline

Homework

Exercise H1.1

1. Use Taylor polynomial approximation to avoid the loss-of-significance errors in the following function formula when x approaches 0:

$$f(x) = \frac{1 - cos(x)}{x^2}$$

2. Reformulate the following function $g(x)$ to avoid the loss-of-significance error in its evaluation for increasing values of x towards $+\infty$:

$$g(x) = x\left(\sqrt{x+1} - \sqrt{x}\right)$$

Exercise H1.2 (From [5])

What is the machine's epsilon for a computer that uses binary arithmetic, $24bits$ for the fraction, and rounds? What if it chops?

Homework

Exercise H1.3 (From [5])

We can compute e^{-x} using Taylor polynomials in two ways, either using:

$$e^{-x} \approx 1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 + \cdots$$

or using

$$e^{-x} = \frac{1}{e^x} \approx \frac{1}{1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \cdots}$$

Discuss which approach is the more accurate. Which one is less susceptible to rounding error?

Homework

Exercise H1.4 (From [2])

Consider the following command sequence:

```
x = 1e-15
y = ((1+x)-1))/x
```

1. What can you say about y? Give reason for the result through the direct analysis of errors propagation in the sequence.
2. Explain the obtained results by calculating the conditioning of the problem.