Rounding
- floating point system is discrete!
  - not all real numbers representable
  - those that are called "machine numbers"
  - others must be *rounded*

```
x  <-- fl(x)
```

- leads to "rounding error" or "roundoff error"

- How to round?
1. Chop - truncate digits - "round to zero"
2. Round to nearest
   - in case of tie go to even

```
Example: Rounding
-----------------
```

Number	Chop	Round to nearest
1.649	1.6	1.6
1.650	1.6	1.6 (tie - round to even)
1.651	1.6	1.7
1.699	1.6	1.7
1.749	1.7	1.7
1.750	1.7	1.8 (tie - round to even)
1.751	1.7	1.8
1.799	1.7	1.8

EXAMPLE : !!!!! warning: don't compare fp numbers with == !!!!!

```
>> 4/3-1 == 1/3
ans = 0

>> single((4/3-1))==single(1/3)
ans = 1

>> (4/3-1)-1/3
ans = -5.5511e-17
```

right way to compare:

```
>> abs((4/3 - 1) - 1/3) <= 1e-16
ans = 1
```

Machine Precision

```
     eps_mach
```

- characterizes accuracy
- "machine epsilon", "machine precision", "unit roundoff"
- depends on rounding rule

```
   _ . _ _ _ ...   _   x x x ... x
   0 . 1 2 3 ... (p-1) p ...
```

- chop: (chop everything at and after b^-p position)
  `b^-(p-1) = b^(1-p)`
- round: (lose up to half of chop)
  `1/2 b^(1-p)`

- `eps_mach` tells us the max possible relative error in representation

```
   | fl(x) - x |
   ------------- <=   eps_mach
       | x |
```

- check:

```
    <= eps_mach * b^e / | x |
     = eps_mach * b^e / (m * b^e)
     = eps_mach / m
    <= eps_mach
```

- alternative characterization

fl(1 + eps_mach) > 1

Examples:
- (Ex. 1) eps_mach (chop, nearest) = .25, .125
- IEEE SP eps_mach (nearest) = 2^-24 ~= 10^-7 (about 7 decimal digits of precision)
- IEEE DP eps_mach (nearest) = 2^-53 ~= 10^-16 (about 16 decimal digits of precision)

Floating Point Math

- adding or subtracting
  - match exponents first
  - must shift smaller number

- if the sum (or diff) contains more than p digits, then the ones smaller than p will be lost
- smallest number may be lost completely

- multiplication ok
  - mult mantissas and sum exponents
  - still need to round though, because product will generally have more digits (up to 2p)

Example

```
    1.23 * 10^5
+   1.00 * 10^4 (10^3, 10^2)
```

- can also get overflow or underflow
- underflow often ok - 0 is good approximation
- overflow more serious problem - can't approximate the number in question

- IEEE standard gives us

```
x flop y = fl(x op y)
```

as long as overflow doesn't occur
- + and * commutative but *not* associative
- Ex: for eps < eps_mach, and 2 eps > eps_mach

```
 ( 1 + eps ) + eps  = 1
   1 + ( eps + eps )  = 1 + 2 eps > 1
```

Rounding Error Analysis

Basic idea is:

```
    fl(x op y) = (x op y)(1 + delta),
        |delta| <= eps_mach, and op = +, -, *, /
```

rearranging, get bound on relative *forward error*:

```
|fl(x op y) - (x op y)|
-----------------------  = |delta| <= eps_mach
     |(x op y)|
```

or, can interpret in terms of *backward error* (with op = +):

```
fl(x + y) = (x + y)(1 + delta) = x(1+delta) + y(1+delta)
```

Example: Compute x(y+z)

```
fl(y+z) = (y+z)(1+d1), |d1|<=eps_mach
and
fl(x(y+z)) = (x(y+z)(1+d1))(1+d2), |d2|<=eps_mach
           = x(y+z)(1+d1+d2+d1d2)
          ~= x(y+z)(1+d1+d2)
           = x(y+z)(1+d), |d| = |d1 + d2| <= 2 eps_mach
```

- pessimistic bound
- typical, multiples of eps_mach accumulate
  - but in practice this is generally ok

Cancellation

problems can arise when subtracting two very close numbers
- result is exactly representable, but
- e.g., if the numbers differ by rounding error, this can basically leave rounding error only after subtracting

Examples

```
  x = 1.92403 * 10^2
- y = 1.92275 * 10^2
  ------------------
      0.00128 * 10^2  = .128 = 1.28 * 10^-1
```

- only 3 significant digits in the result

BAD: computing *small quantity* as a difference of *large quantities*

```
e^x = 1 + x + x^2/2 + x^3/3! + ..., for x < 0
```

Example: Quadratic formula

```
ax^2 + bx + c = 0

    -b +- sqrt(b^2-4ac)
b = -------------------
            2a

0.05010 x^2 - 98.78 x + 5.015
roots ~= 1971.605916,   answer to 10 digits
         0.05077069387

b^2 - 4ac = 9757-1.005 = 9756  answer to 4 digits
sqrt( " ) = 98.77
roots: (98.78 +- 98.77) / 0.1002 = 1972, 0.09980
```

subtraction of two *close* numbers (cancellation error), followed by division by *small* number (amplification)