Loss of Accuracy

**Addition and Subtraction** might necessitate mantissa shift to make exponents match. This can cause the loss of some (or all) digits in one operand.

**Multiplication** The product of two $n$-digit numbers is a $2n$-digit number. Again, digits are lost, the product might not even be representable.

**Division** A quotient might contain more (even infinitely many) digits than the dividend/divisor. (c.f. $\frac{2}{3} = .666 \ldots$)

The result of floating point operations might differ from the floating point representation of the result of the operations on real numbers.

**Caveat:** Machine numbers do not form a ring!
Cancellation

Subtraction of two numbers of similar magnitude gives a result that might have a smaller exponent and thus loses significant digits

\[ 1.2345678 \cdot 10^5 - 1.2345663 \cdot 10^5 = 1.5 \cdot 10^{-1} \]

Digits lost to cancellation are the most significant digits, digits lost by rounding are the least significant.

Thus: It is a bad idea to compute a small quantity as difference of large quantities.
Example

When solving a quadratic equation, the expression $b^2 - 4ac$ occurs. Consider $b = 3.34$, $a = 1.22$ and $c = 2.28$. Then $b^2 - 4ac = .0292$.

With FPN ($\beta = 10$, $p = 3$) $b^2$ rounds to 11.2 and $4ac$ to 11.1. The final answer thus is .1, an error of 70 ulp. This despite the fact that the final subtraction is exact.

Sometimes it is possible to rearrange expressions to avoid bad cancellation. For example consider the formula for the root of a quadratic equation

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

in the case that $b^2 \gg ac$. Then $b^2 - 4ac$ does not involve any cancellation but one of the sums $-b \pm \sqrt{b^2 - 4ac}$ can.
To avoid this, multiply numerator and denominator with 
\(-b - \sqrt{b^2 - 4ac}\) to get

\[
r = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}
\]

with changed sign \(\pm \rightarrow \mp\). Then use one formula for one root each to avoid cancellation.

Similarly it can be better to evaluate \(x^2 - y^2\) as \((x + y)(x - y)\).
Polynomial Evaluation

Another common example is the evaluation of polynomials. To evaluate

$$\sum_{i=0}^{n} a_i x^i$$

one evaluates

$$a_0 + x (a_1 + x (a_2 + x (a_3 + x (\cdots))))$$

which requires fewer operations (and thus less loss of accuracy).
Conditioning

A problem is called *ill-conditioned* (opposite: *well-conditioned*) if a relative change in the input can produce a much larger relative change in the solution.

The *condition number* of a problem is

\[
\text{cond} = \left| \frac{\text{relative change in solution}}{\text{relative change in input}} \right|
\]

A problem is ill-conditioned, if \(\text{cond} \gg 1\).

Similarly: *stable* algorithm.
Example: Evaluating functions

Evaluate function $f$ for approximate input value $\hat{x} = x + h$ instead of true input value $x$:

absolute error: $f(x + h) - f(x) \approx f'(x) \cdot h$

relative error: $\frac{f(x + h) - f(x)}{f(x)} \approx \frac{f'(x) \cdot h}{f(x)}$

$\text{cond} \approx \left| \frac{hf'(x)/f(x)}{h/x} \right| = \left| x \frac{f'(x)}{f(x)} \right|.$

For $f(x) = \cos(x)$, $x \to \frac{\pi}{2}$: $\text{cond} \to \infty$
Backward Error Analysis

Suppose we compute a value $f(x)$ approximately as $\hat{f}(x)$.

“Forward” Error: $|\hat{f}(x) - f(x)|$.

We now want to find out what error in the input could have caused such an output discrepancy.

(If it is comparable to the error of the input data (or roundoff error), the calculation is well-behaved.)

Try to find $\hat{x}$ such that $f(\hat{x}) = \hat{f}(x)$.

“Backward” Error: $|\hat{x} - x|$.
Exceptions

The IEEE standard defines special values:

\textbf{Inf} infinity as a result of finite number divided by zero.

\textbf{NaN} not a number: \(0/0, 0 \times \infty, \infty/\infty\), etc.

Both are represented by reserved values in the exponent field.
Programming with Floating Point Numbers

When programming, we have to specify which format of floating point number to use:

- **IEEE Precision**: C, C++, FORTRAN
- single: float
  - REAL or REAL*4
- double: double
  - DOUBLE PRECISION or REAL*8
- double extended: long double
  - REAL*16 (not all)

For example:

```c
void main()
{
    float a;
    a=3.1415926;
    printf("%f\n", a);
}
```
To print a floating point number, we use `printf`. The first argument is a string, in which expressions starting with a `%` indicate the format for numbers:

- `%f` float (display 6 digits), works also for double
- `%e` same but display in exponent form
- `%15f` float (display 15 digits)
- `%Lf` long double
- `%12Le` long double, exponent form, 12 digits

There are also conventions on how to convert numbers of different types when performing arithmetic with them together. Typically they state, that arguments are converted in the “more accurate” type first.
However this sometimes might be still misleading:

```c
float a;
a=10/3;
```

produces the result $a = 3$ (division among integers!). To force floating point evaluation, one can write:

```c
a=10./3;
```
Let \( f : \mathbb{R} \rightarrow \mathbb{R} \) be a function that is infinitely often differentiable at \( x_0 \in \mathbb{R} \). Then the Taylor series gives for \( x \in \mathbb{R} \) that

\[
f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + \frac{f^{(4)}(x_0)}{4!}(x - x_0)^4 + \cdots
\]

\[
= \sum_{i=0}^{\infty} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i
\]
For \( x \) in the neighborhood of \( x_0 \), one can use partial sums as approximation:

\[
 f(x) \approx f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n =: \mathcal{T}_{f,x_0,n}(x)
\]
The *remainder* of this approximation is

\[ R_{n+1} = f(x) - T_{f,x_0,n}(x) \]

We also have that there is an \( \xi \in (x_0, x) \) such that

\[ R_{n+1} = \frac{f^{(n+1)}(\xi)}{(n + 1)!} (x - x_0)^{n+1} \]

which gives the estimate

\[ |R_{n+1}| < \sup_{\xi \in (x_0, x)} \left| f^{(n+1)}(\xi) \right| \frac{|x - x_0|^{n+1}}{(n + 1)!} \]
For example

For \( x_0 = 0 \) we have

\[
\sin(x) \approx \sin(x_0) + \frac{\cos(x_0)}{1!}(x - x_0) - \frac{\sin(x_0)}{2!}(x - x_0)^2 - \frac{\cos(x_0)}{3!}(x - x_0)^3 + \frac{\sin(x_0)}{4!}(x - x_0)^4
\]

\[
= (x - x_0) - \frac{1}{6}(x - x_0)^3.
\]

with an error estimate

\[
R_5 = \frac{\cos(\xi)}{120}(x - x_0)^5, \quad |R_5| < \frac{1}{120}(x - x_0)^5
\]