# Floating-Point Arithmetic

## 1 Floating-Point Numbers

Computer numbers:

$$\pm d_0.d_1 \cdots d_{p-1} \times \beta^e$$

$\beta$: base,
$e$: exponent,
$p$: precision.
IEEE standard

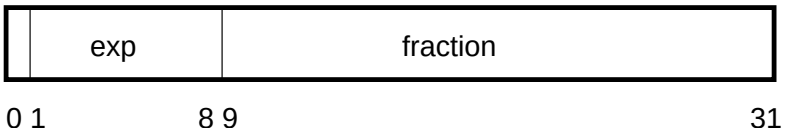


Figure 1: IEEE single precision floating-point word format.

**Four parameters**

Base $\beta$:

- IEEE standard: 2
- Other commonly used: 10, 16
  Base 16 provides wider range, but wastes bits in fraction. For example, $1.0 = .00010000_2 \times 16^1$.

Exponents $e_{\min}$ and $e_{\max}$

- IEEE standard: $e_{\min} = -126$, $e_{\max} = 127$
  biased representation, bias 127.
- Overflow and underflow
  when the exponent is too large or too small.

  **Example 1** *In IEEE single precision, if* $a = 3.0 \times 10^{-30}$*, then* $a * a$ *underflows, usually set to zero. If* $a = 3.0 \times 10^{30}$*,* $a * a$ *overflows, set to* $+\infty$.

- The infinity symbol is represented by exponent $e_{\max} + 1$ and a zero fraction.
- Usual mathematical convention for infinity

$$\infty + \infty = \infty, \qquad (\text{finite})/\infty = 0.$$

- Avoiding unnecessary underflow and overflow by scaling.

**Example 2** *Given* $x = (a, b)^T$, $a = 1.0 \times 10^{30}$, $b = 1.0$, *compute* $c = \|x\|_2 = \sqrt{a^2 + b^2}$.

*scaling:* $s = \max\{|a|, |b|\} = 1.0 \times 10^{30}$

$a \leftarrow a/s$ *(*$1.0$*),*
$b \leftarrow b/s$ *(*$1.0 \times 10^{-30}$*)*
$t = \sqrt{a * a + b * b}$ *(*$1.0$*)*
$c \leftarrow t * s$ *(*$1.0 \times 10^{30}$*)*

Study the following BLAS code for its robustness, accuracy, and efficiency.

```
      DOUBLE PRECISION FUNCTION DNRM2 ( N, X, INCX )
*     .. Scalar Arguments ..
      INTEGER                          INCX, N
*     .. Array Arguments ..
      DOUBLE PRECISION                 X( * )
*     ..
*
*  DNRM2 returns the euclidean norm of a vector via the function
*  name, so that
*
*     DNRM2 := sqrt( x'*x )
*
*
*
*  -- This version written on 25-October-1982.
*     Modified on 14-October-1993 to inline the call to DLASSQ.
*     Sven Hammarling, Nag Ltd.
*
*
*     .. Parameters ..
```

```
      DOUBLE PRECISION      ONE         , ZERO
      PARAMETER           ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*     .. Local Scalars ..
      INTEGER               IX
      DOUBLE PRECISION      ABSXI, NORM, SCALE, SSQ
*     .. Intrinsic Functions ..
      INTRINSIC             ABS, SQRT
*     ..
*     .. Executable Statements ..
      IF( N.LT.1 .OR. INCX.LT.1 )THEN
         NORM  = ZERO
      ELSE IF( N.EQ.1 )THEN
         NORM  = ABS( X( 1 ) )
      ELSE
         SCALE = ZERO
         SSQ   = ONE
*        The following loop is equivalent to this call to the
*        LAPACK auxiliary routine:
*        CALL DLASSQ( N, X, INCX, SCALE, SSQ )
*
         DO 10, IX = 1, 1 + ( N - 1 )*INCX, INCX
            IF( X( IX ).NE.ZERO )THEN
               ABSXI = ABS( X( IX ) )
               IF( SCALE.LT.ABSXI )THEN
                  SSQ   = ONE   + SSQ*( SCALE/ABSXI )**2
                  SCALE = ABSXI
               ELSE
                  SSQ   = SSQ   +     ( ABSXI/SCALE )**2
               END IF
            END IF
   10    CONTINUE
         NORM  = SCALE * SQRT( SSQ )
      END IF
*
      DNRM2 = NORM
      RETURN
*
*     End of DNRM2.
*
      END
```

| Operation | NaN Produced By |
|---|---|
| $+$ | $\infty + (-\infty)$ |
| $*$ | $0 * \infty$ |
| $/$ | $0/0$, $\infty/\infty$ |
| REM | $x$ REM 0, $\infty$ REM $y$ |
| sqrt | sqrt($x$) when $x < 0$ |

Table 1: Operations that Produce a NaN

- What is $\infty + (-\infty)$?
  NaN, Not a Number

Precision

- IEEE standard, single precision: $p = 24$, one hidden bit.
- Rounding error
  Due to finite precision arithmetic, a computed result must be rounded to fit storage format.

  **Example 3** $\beta = 10$, $p = 4$

  $a = 1.234 \times 10$, $b = 3.156 \times 10^{-1}$

  $x = a + b = 1.26556 \times 10^1$ *(exact)*

  $\hat{x} = \mathrm{fl}(a + b) = 1.266 \times 10^1$

  *the result rounded to the nearest computer number. The relative error*

  $$\frac{|\hat{x} - x|}{|x|} \approx 0.35 \times 10^{-3}.$$

- In general,
  $$\frac{|\hat{x} - x|}{|x|} \leq \frac{1}{2}\beta^{-p+1}.$$
- Unit of roundoff
  $$u = \frac{1}{2}\beta^{-p+1}.$$
  When $\beta = 2$, $u = 2^{-p}$.

- Computing $u$: the largest $\epsilon$ such that $\mathrm{fl}(1.0+\epsilon) = 1.0$.

```
eps = 1.0;
while (1.0 + eps) != 1.0
    eps = eps/2;
return eps
```

Some compilers optimize the test `(1.0 + eps) != 1.0` to `eps != 0`.

- Denormals are represented by exponent $e_{\min} - 1$ and a nonzero fraction.
- Zeros are represented by exponent $e_{\min} - 1$ and a zero fraction.
- Correctly rounded operations
  The computed result must be the same as if it were computed exactly and then rounded, usually to the nearest floating-point number.

  **Example 4**

  $$a \oplus b = \mathrm{fl}(a+b) = (a+b)(1+\epsilon) \quad |\epsilon| \le u,$$

  *for operations* $+$, $-$, $*$, $/$, $\sqrt{\ }$.

- Floating-point operations are commutative, but not associative, not distributive. (Examples?)

# 2 Backward Error Analysis

We can write

$$a \oplus b = \mathrm{fl}(a+b) = (a+b)(1+\epsilon) = a(1+\epsilon) + b(1+\epsilon) \quad |\epsilon| \le u.$$

The computed result is the exact result with slightly perturbed data $a(1+\epsilon)$ and $b(1+\epsilon)$.

**Example 5** *Computing the sum:*

$$s_n = \mathrm{fl}(x_1 + x_2 + \cdots + x_n)$$

*Denote the partial sum* $s_i = \mathrm{fl}(x_1 + x_2 + \cdots + x_i)$, *then*

$$
\begin{aligned}
s_2 &= \mathrm{fl}(x_1 + x_2) = x_1(1+\epsilon_1) + x_2(1+\epsilon_1) \\
s_3 &= \mathrm{fl}(s_2 + x_3) = (s_2 + x_3)(1+\epsilon_2) \\
&= x_1(1+\epsilon_1)(1+\epsilon_2) + x_2(1+\epsilon_1)(1+\epsilon_2) \\
&+ x_3(1+\epsilon_2) \\
s_n &= x_1(1+\epsilon_1)(1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\
&+ x_2(1+\epsilon_1)(1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\
&+ x_3(1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\
&+ \cdots \\
&+ x_{n-1}(1+\epsilon_{n-2})(1+\epsilon_{n-1}) \\
&+ x_n(1+\epsilon_{n-1})
\end{aligned}
$$

*Define*

$$
\begin{aligned}
1+\eta_1 &= (1+\epsilon_1)(1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\
1+\eta_2 &= (1+\epsilon_1)(1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\
1+\eta_3 &= (1+\epsilon_2)\cdots(1+\epsilon_{n-1}) \\
&\cdots \\
1+\eta_{n-1} &= (1+\epsilon_{n-2})(1+\epsilon_{n-1}) \\
1+\eta_n &= (1+\epsilon_{n-1})
\end{aligned}
$$

$$
\begin{aligned}
|\eta_n| &= |\epsilon_{n-1}| \le u \\
1+\eta_{n-1} &= 1 + (\epsilon_{n-2} + \epsilon_{n-1}) + \epsilon_{n-2}\epsilon_{n-1} \\
|\eta_{n-1}| &\approx |\epsilon_{n-2} + \epsilon_{n-1}| \le 2u
\end{aligned}
$$

*In general,*

$$
\begin{aligned}
|\eta_1| &\le (n-1)u \\
|\eta_i| &\le (n-i+1)u, \quad i = 2, 3, ..., n
\end{aligned}
$$

*A more rigorous bound: If* $nu \le 0.1$ *and* $|\epsilon_i| \le u$ *(*$i = 1, 2, ..., n$*), then*

$$(1+\epsilon_1)(1+\epsilon_2)\cdots(1+\epsilon_n) = 1+\eta$$

*where*

$$|\eta| \le 1.06nu.$$

*Thus*

$$\text{fl}(x_1 + \cdots x_n) = x_1(1+\eta_1) + \cdots + x_n(1+\eta_n)$$

$|\eta_1| \leq 1.06(n-1)u$
$|\eta_i| \leq 1.06(n-i+1)u,\ i = 2, 3, ..., n$

How bad is the accumulation error? Using double precision, $u = 10^{-15}$, for $nu \geq 0.1$, $n \geq 10^{14}$. How long does it take to perform $10^{14}$ additions?

Backward error analysis
The computed result $(fl(x_1 + \cdots + x_n))$ is the exact result of the problem with slightly perturbed data $(x_1(1+\eta_1), ..., x_n(1+\eta_n))$.

# 3 Perturbation Analysis

Disregard rounding errors and consider the perturbed data

$$\tilde{x}_i = x_i(1+\epsilon_i) \qquad |\epsilon_i| \leq \epsilon,$$

in the sum

$$s = x_1 + x_2 + \cdots x_n,$$

we have

$$\tilde{s} = \tilde{x}_1 + \tilde{x}_2 + \cdots + \tilde{x}_n$$

and

$$\frac{|\tilde{s} - s|}{|s|} \leq \frac{|x_1| + |x_2| + \cdots + |x_n|}{|x_1 + x_2 + \cdots x_n|}\,\epsilon.$$

Set

$$\kappa = \frac{|x_1| + |x_2| + \cdots + |x_n|}{|x_1 + x_2 + \cdots x_n|},$$

then

$$\frac{|\tilde{s} - s|}{|s|} \leq \kappa\,\epsilon.$$

The number $\kappa$ tells how the relative errors in the data are magnified in the relative error in the result. This is called the condition number for the problem. In the above example, $\kappa \geq 1$ and $\kappa = 1$ when $x_i \geq 0$.

In general, we can view a problem with data $\mathbf{x}$ and result $\mathbf{y}$ as a function $\mathbf{y} = f(\mathbf{x})$. The result of the perturbed problem is $\breve{\mathbf{y}} = f(\mathbf{x} + \Delta\mathbf{x})$. The sensitivity is measured by

$$\text{cond} = \frac{|\breve{\mathbf{y}} - \mathbf{y}|/|\mathbf{y}|}{|\Delta\mathbf{x}|/|\mathbf{x}|} = \frac{|f(\mathbf{x} + \Delta\mathbf{x}) - f(\mathbf{x})|}{|\Delta\mathbf{x}|} \frac{|\mathbf{x}|}{|f(\mathbf{x})|} \approx |f'(\mathbf{x})| \frac{|\mathbf{x}|}{|f(\mathbf{x})|}.$$

Note that the conditioning of a problem is independent of rounding errors and algorithms for solving the problem.

The power of the backward error analysis

- Separates the properties of the problem to be computed and those of the algorithm used;
- Ill-conditioned problem:
  small perturbations on data can cause large errors in the solution;
- Stable algorithm:
  the computed solution is the exact solution of the problem with slightly perturbed data. If the perturbation is smaller than the measurement errors in the data, can't blame computer for large error in the result.

Does a well-conditioned problem always lead to good results (regardless of the algorithm used)?

Does a stable algorithm always compute accurate solution (regardless of the conditioning of the problem)?

**Summary**

- A computer number system is determined by four parameters: Base, precision, $e_{\min}$, and $e_{\max}$
- IEEE floating-point standards, single precision and double precision. Special values: Denormals, $\pm\infty$, NaN, $\pm 0$, and their binary representations.
- Error measurements: Absolute and relative errors, unit of roundoff
- Issues in floating-point computation: Overflow, underflow, cancellation

- Error analysis: Forward and backward errors, sensitivity of a problem and stability of an algorithm

### Reading Material

George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, Inc., 1977. Chapter 2.

D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, **23**(1) (1991), 5–48.

Nicholas J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, 2002. Chapters 1 and 2.