Machine Numbers and Machine Arithmetic

A Matlab program such as

```
x=.1; y1=cos(x); y=1-y1
```

is not evaluated exactly. We can only store a certain number of digits for each number. Instead of arbitrary real numbers we only have finitely many **machine numbers** available. Arithmetic operations like `z=x+y` or `s=sqrt(x)` are not performed exactly, but give a result which is again a machine number. This is called **machine arithmetic**.

We want to

- represent real numbers with a **large range of magnitudes**, e.g., 10^{-100} to 10^{100}
- achieve **small relative errors**: rounding a number to the closest machine should give a relative error of at most $\varepsilon_M \approx 10^{-16}$.

Simple base 10 machine numbers

Some machines (e.g. all calculators) use base 10 machine numbers. In decimal notation we have e.g.

$$(.341)_{10} = 3 \cdot 10^{-1} + 4 \cdot 10^{-2} + 1 \cdot 10^{-3}.$$

In general an n-digit base 10 number with digits $d_j \in \{0,\dots,9\}$ is

$$(.d_1 d_2 \dots d_n)_{10} = d_1 \cdot 10^{-1} + d_2 \cdot 10^{-2} + \cdots + d_n \cdot 10^{-n}.$$

We can write a number $x \in \mathbb{R}$ in the form $x = \pm q \cdot 10^e$ with a **mantissa** q and an **exponent** e. E.g., the number $x = 12345$ can be written as

$$x = 12345 = .12345 \cdot 10^5 = .012345 \cdot 10^6 = .0012345 \cdot 10^7$$

We call the first form $.12345 \cdot 10^5$ the **normalized** representation since the first digit d_1 after the decimal point is nonzero.

Any number $x \in \mathbb{R}$ with $x \neq 0$ can be written as

$$x = \pm q \cdot 10^e, \qquad \frac{1}{10} \leq q < 1, \qquad e \in \mathbb{Z} \tag{1}$$

For machine numbers we want to represent the mantissa with n digits, and use a range $e_{\min} \leq e \leq e_{\max}$ of exponents.
Simple base 10 machine numbers are either **normalized numbers** or **zero**:

$$\hat{x} = \begin{cases} \pm(.d_1 d_2 \dots d_n)_{10} \cdot 10^e, & d_j \in \{0,\dots,9\}, \quad d_1 \neq 0, \qquad e \in \mathbb{Z}, \quad e_{\min} \leq e \leq e_{\max} \\ 0 \end{cases}$$

The **largest machine number** is $x_{\max} = (.99\cdots 9)_{10} \cdot 10^{e_{\max}} = (1 - 10^{-n}) \cdot 10^{e_{\max}}$,

the **smallest positive machine number** is $x_{\min} = (.10\dots 0)_{10} \cdot 10^{e_{\min}} = 10^{e_{\min}-1}$.

For calculators we have typically $n = 8$ mantissa digits, and can use exponents between $e_{\min} = -99$ and $e_{\max} = 99$.

Rounding: A given number $x \in \mathbb{R}$ is represented by a machine number $\hat{x}$. This operation is denoted by $fl(x)$ ("floating point approximation").

- Write x in the form $x = \pm q \cdot 10^e$ with $\frac{1}{10} \leq q < 1$ and $e \in \mathbb{Z}$
- If $e_{\min} \leq e \leq e_{\max}$: Find the nearest mantissa $\hat{q} = (.d_1 d_2 \dots d_n)_{10}$ to q, then $\hat{x} = \pm \hat{q} \cdot 10^e$
- If $e > e_{\max}$: "**Overflow**" , i.e., $|x|$ is too large (we will explain later what to do in this case)
- If $e < e_{\min}$: "**Underflow**", let $\hat{x}$ be 0 or x_{min}, whatever is closer

Example: Assume we have a machine with $n = 3$, $e_{\min} = -99$ and $e_{\max} = 99$. We want to find $\hat{x} = fl(x)$ for $x = \dfrac{2}{300}$.

- $x = +\frac{2}{3} \cdot 10^{-2}$, i.e., $q = \frac{2}{3}$ and $e = -2$. Note that $e \in [e_{\min}, e_{\max}]$ so we don't have overflow or underflow.
- Now we need to approximate the mantissa $q = \frac{2}{3} = (.666666\ldots)_{10}$ by a number $\hat{q} = (.d_1 d_2 d_3)_{10}$.
  The closest number to the left is $\hat{q}_{\text{left}} = (.666)_{10}$, the closest number to the right is $\hat{q}_{\text{right}} = (.667)_{10}$. In order to decide which is closer we look at the midpoint $q_{\text{mid}} = (.6665)_{10}$. If $q < q_{\text{mid}}$ we round down to $\hat{q}_{\text{left}}$, if $q > q_{\text{mid}}$ we round up to $\hat{q}_{\text{right}}$ (if $q = q_{\text{mid}}$ it does not matter which we choose).
  Here $q = (.666666\cdots)_{10} > q_{\text{mid}} = (.666500)_{10}$, therefore $\hat{q} = \hat{q}_{\text{right}}$ and
  $$\hat{x} = fl(x) = +(.667)_{10} \cdot 10^{-2}.$$

Now we want to find an **upper bound for the rounding error**: If we don't have overflow or underflow we have $x = \pm q \cdot 10^e$ and $\hat{x} = \pm \hat{q} \cdot 10^e$. Hence
$$\left| \frac{\hat{x} - x}{x} \right| = \frac{|\hat{q} \cdot 10^e - q \cdot 10^e|}{q \cdot 10^e} = \frac{|\hat{q} - q|}{q}$$
In the denominator we have $q \geq \frac{1}{10}$. In the numerator we have $|\hat{q} - q| \leq \frac{1}{2} \cdot 10^{-n}$ since the spacing between two successive mantissa values is 10^{-n}, and the largest possible value of $|\hat{q} - q|$ is half this distance. Hence the rounding error can be bounded by
$$\left| \frac{\hat{x} - x}{x} \right| = \frac{|\hat{q} - q|}{q} \leq \frac{\frac{1}{2} \cdot 10^{-n}}{1/10} = \frac{1}{2} \cdot 10^{-n+1}$$
This number is called the **machine epsilon:** $\boxed{\varepsilon_M = \frac{1}{2} \cdot 10^{-n+1}}$.

In our example we had $n = 3$, therefore $\varepsilon_M = \frac{1}{2} \cdot 10^{-2} = 5 \cdot 10^{-3}$. For $x = \frac{2}{300}$ we obtained $\hat{x} = .667 \cdot 10^{-2}$, so $\dfrac{\hat{x} - x}{x} \approx 5 \cdot 10^{-4}$.
For $x = 100.4 = (.1004) \cdot 10^3$ we obtain $\hat{x} = (.100) \cdot 10^3 = 100$, so $\dfrac{\hat{x} - x}{x} \approx 4 \cdot 10^{-2}$.

Simple base 2 machine numbers

Most computers use base 2 machine numbers. In binary notation we have e.g.
$$(.101)_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}.$$
In general an n-digit base 2 number with digits $d_j \in \{0, 1\}$ is
$$(.d_1 d_2 \ldots d_n)_2 = d_1 \cdot 2^{-1} + d_2 \cdot 2^{-2} + \cdots + d_n \cdot 2^{-n}.$$

We can write a number $x \in \mathbb{R}$ in the form $x = \pm q \cdot 2^e$ with a **mantissa** q and an **exponent** e. E.g., the number $x = (1101)_2$ can be written as
$$x = (1101)_2 = (.1101)_2 \cdot 2^4 = (.01101)_2 \cdot 2^5 = (.001101)_2 \cdot 2^6$$
We call the first form $(.1101)_2 \cdot 2^4$ the **normalized** representation since the first digit d_1 after the point is nonzero.

Any number $x \in \mathbb{R}$ with $x \neq 0$ can be written as
$$x = \pm q \cdot 2^e, \qquad \frac{1}{2} \leq q < 1, \qquad e \in \mathbb{Z} \tag{2}$$

For machine numbers we want to represent the mantissa with n digits, and use a range $e_{\min} \leq e \leq e_{\max}$ of exponents.
Simple base 2 machine numbers are either **normalized numbers** or **zero**:

$$\boxed{\hat{x} = \begin{cases} \pm(.d_1 d_2 \ldots d_n)_2 \cdot 2^e, & d_j \in \{0, 1\}, \quad d_1 = 1, \qquad e \in \mathbb{Z}, \quad e_{\min} \leq e \leq e_{\max} \\ 0 \end{cases}}$$

Note that for normalized numbers we always have $d_1 = 1$, hence this digit does not have to be stored.
The **largest machine number** is $x_{\max} = (.11\cdots1)_2 \cdot 2^{e_{\max}} = (1 - 2^{-n}) \cdot 2^{e_{\max}}$,

the **smallest positive machine number** is $x_{\min} = (.10\ldots0)_2 \cdot 2^{e_{\min}} = 2^{e_{\min}-1}$.

Rounding: A given number $x \in \mathbb{R}$ is represented by a machine number $\hat{x}$. This operation is denoted by $fl(x)$ ("floating point approximation").

- Write x in the form $x = \pm q \cdot 2^e$ with $\frac{1}{2} \le q < 1$ and $e \in \mathbb{Z}$
- If $e_{\min} \le e \le e_{\max}$: Find the nearest mantissa $\hat{q} = (.d_1 d_2 \dots d_n)_2$ to q, then $\hat{x} = \pm \hat{q} \cdot 2^e$
- If $e > e_{\max}$: "**Overflow**" , i.e., $|x|$ is too large (we will explain later what to do in this case)
- If $e < e_{\min}$: "**Underflow**", let $\hat{x}$ be 0 or x_{min}, whatever is closer

Example: What happens when the Matlab command `x=.1` is executed? Matlab uses binary machine numbers with $n = 53$, $e_{\min} = -1021$, $e_{\max} = 1024$. We want to find $\hat{x} = fl(x)$ for $x = \dfrac{1}{10}$.

- $x = +\frac{8}{10} \cdot 2^{-3}$, i.e., $q = \frac{8}{10}$ and $e = -3$. Note that $e \in [e_{\min}, e_{\max}]$ so we don't have overflow or underflow.
- Now we need to approximate the mantissa $q = \frac{8}{10}$ by a number $\hat{q} = (.d_1 d_2 \dots d_{53})_2$. Note that we have in base 2 (digits after d_{53} are shown in red)

$$
\begin{aligned}
q &= (.1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1\textcolor{red}{10011\dots})_2 \\
\hat{q}_{\text{left}} &= (.1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1)_2 \\
\hat{q}_{\text{right}} &= (.1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1101\,0)_2 \\
q_{\text{mid}} &= (.1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1100\,1\textcolor{red}{10000})_2
\end{aligned}
$$

  The closest number to the left is $\hat{q}_{\text{left}}$, the closest number to the right is $\hat{q}_{\text{right}}$. In order to decide which is closer we look at the midpoint q_{mid}. If $q < q_{\text{mid}}$ we round down to $\hat{q}_{\text{left}}$, if $q > q_{\text{mid}}$ we round up to $\hat{q}_{\text{right}}$ (if $q = q_{\text{mid}}$ it does not matter which we choose).
  Here $q > q_{\text{mid}}$, therefore $\hat{q} = \hat{q}_{\text{right}}$ and

$$\hat{x} = fl(x) = +\hat{q}_{\text{right}} \cdot 2^{-3}$$

Now we want to find an **upper bound for the rounding error**: If we don't have overflow or underflow we have $x = \pm q \cdot 2^e$ and $\hat{x} = \pm \hat{q} \cdot 2^e$. Hence

$$\left| \frac{\hat{x} - x}{x} \right| = \frac{|\hat{q} \cdot 2^e - q \cdot 2^e|}{q \cdot 2^e} = \frac{|\hat{q} - q|}{q}$$

In the denominator we have $q \ge \frac{1}{2}$. In the numerator we have $|\hat{q} - q| \le \frac{1}{2} \cdot 2^{-n}$ since the spacing between two successive mantissa values is 2^{-n}, and the largest possible value of $|\hat{q} - q|$ is half this distance. Hence the rounding error can be bounded by

$$\left| \frac{\hat{x} - x}{x} \right| = \frac{|\hat{q} - q|}{q} \le \frac{\frac{1}{2} \cdot 2^{-n}}{1/2} = \tfrac{1}{2} \cdot 2^{-n+1} = 2^{-n}$$

This number is called the **machine epsilon:** $\boxed{\varepsilon_M = 2^{-n}}$.

In Matlab we have $n = 53$, therefore $\varepsilon_M = 2^{-53} \approx 1.11 \cdot 10^{-16}$. In our example with $x = \frac{1}{10}$ we have $\dfrac{\hat{x} - x}{x} = \dfrac{\hat{q} - q}{q} \approx 5.55 \cdot 10^{-17}$.

IEEE754 machine numbers

Our "simple base 2 machine numbers" have some problems.

There is a huge hole around 0

The distance between 0 and the smallest positive machine number $x_{\min}$ is much larger than the distance between $x_{\min}$ and the next larger machine number $x_1 := x_{\min}(1 + 2^{-n})$:

$$x_{\min} - 0 = 2^{e_{\min}} \gg x_1 - x_{\min} = 2^{-n} \cdot 2^{e_{\min}}$$

This has unpleasant consequences:

- Rounding numbers x with $|x| > x_{\min}$ causes a relative error of size$\leq \varepsilon_M$. Rounding numbers x with $|x| < x_{\min}$ gives either 0 or $x_{\min}$ and causes a relative error of size$\leq$ 100% (underflow). If a program generates values slightly smaller than $x_{\min}$ the accuracy decreases dramatically.
- The two statements **`if y>x`** and **`if y-x>0`** have different meanings: For the machine numbers $x = x_{\min}$ and $y = x_1$ the expression **`y>x`** evaluates to **`true`** since x_1 is a larger machine number than $x_{\min}$. But the expression **`y-x>0`** evaluates to **`false`**: The machine first computes $y - x = x_1 - x_{\min} = 2^{-n} \cdot 2^{e_{\min}}$, then this value is rounded to the closest machine number which is 0.

We can fix this by filling in the hole around 0: So far the smallest positive numbers are obtained with $e = e_{\min}$, they have a spacing of $2^{-n} \cdot 2^{e_{\min}}$:

$$\pm(.1d_2 \dots d_n)_2 \cdot 2^{e_{\min}}, \qquad d_j \in \{0,1\}$$

We now add the "**subnormal numbers**" which have the same spacing of $2^{-n} \cdot 2^{e_{\min}}$:

$$\pm(.0d_2 \dots d_n)_2 \cdot 2^{e_{\min}}, \qquad d_j \in \{0,1\}$$

Note that these values include the **two distinct machine numbers** $+0$ **and** -0 (using signs "+" and "−" with $d_2 = \cdots = d_n = 0$). We will explain below that this is a feature, not a bug.

Now rounding a number x with $|x| \leq x_{\max}$ is more well-behaved since the finest spacing $2^{-n} \cdot 2^{e_{\min}}$ is used around 0: We get $\hat{x} = fl(x)$ with

$$\left| \frac{\hat{x} - x}{x} \right| \leq \begin{cases} 2^{-n} & \text{for } |x| \geq x_{\min} \\ \min\left\{2^{-n} \frac{x_{\min}}{x}, 1\right\} & \text{for } |x| < x_{\min} \end{cases}$$

This is called "**gradual underflow**": If we generate values slightly smaller than $x_{\min}$ the rounding error only increases slightly (instead of jumping to 100%).

We need to specify what happens for overflow, division by 0, 0/0 etc.

We introduce special values **`+Inf`**, **`-Inf`** for handling overflow:

```
>> x=1e300; -x*x
ans =
  -Inf
```

We can perform arithmetic with `Inf` and `-Inf`: E.g., `5-Inf` gives `-Inf`, `Inf*Inf` gives `Inf`, `0/Inf` gives `0` etc.

Note that there are actually two distinct machine numbers `+0` and `-0`. These are both displayed as 0, and the comparison `+0==-0` is defined as true. But these two values can preserve the sign information in the case of an underflow:

```
>> x=1e-300
x =
                    1e-300
>> y=-x*x                      % underflow to -0, displayed as 0
y =
     0
>> 1/y                         % 1/-0 gives -Inf
ans =
  -Inf
```

For indeterminite expressions like `Inf-Inf`, `0*Inf` or `0/0` we introduce the special value **NaN ("Not a Number")**. This value is also useful for representing a missing data value. Arithmetic operations involving `NaN` give again `NaN` (with a few exceptions).

Summary:

IEEE754 machine numbers have the following form with $d_j \in \{0,1\}$ and integer e:

$$\hat{x} = \begin{cases} \pm(.1d_2 \dots d_n)_2 \cdot 2^e, \quad e_{\min} \le e \le e_{\max} & \text{(normalized numbers)} \\ \pm(.0d_2 \dots d_n)_2 \cdot 2^{e_{\min}} & \text{(subnormal numbers, includes } +0, -0\text{)} \\ \text{Inf, -Inf, NaN} & \text{(special values)} \end{cases}$$

Single Precision numbers (type `float` in C) use 4 bytes = 32 bits (1 for sign, 8 for exponent, 23 for mantissa).

Double Precision numbers (type `double` in C) use 8 bytes = 64 bits (1 for sign, 11 for exponent, 52 for mantissa).

	bits	n	$e_{\max}$	$e_{\min}$	ε_M	$x_{\max}$	$x_{\min}$
Single Precision	32	24	128	-125	$2^{-24} \approx 6 \cdot 10^{-8}$	$\approx 2^{128} \approx 3 \cdot 10^{38}$	$2^{-126} \approx 10^{-38}$
Double Precision	64	53	1024	-1021	$2^{-53} \approx 10^{-16}$	$\approx 2^{1024} \approx 2 \cdot 10^{308}$	$2^{-1022} \approx 2 \cdot 10^{-308}$

Note that there are subnormal numbers smaller than $x_{\min}$ available. The smallest positive subnormal number is $2^{-n}2^{e_{\min}} = 2^{-1074} \approx 5 \cdot 10^{-324}$ for double precision. But the value $x_{\min} \approx 2 \cdot 10^{-308}$ is important since **values $|x| < x_{\min}$ can cause roundoff errors larger than ε_M.**

Machine arithmetic

Our machine has built-in operations (like $x+y$, x/y, $\sqrt{x}$, $\sin x$) which operate on machine numbers (this includes $+0$, -0, Inf, $-$Inf, NaN).

Note that for machine numbers x, y the value $x+y$ is usually not a machine number. E.g. for decimal machine numbers with $n = 3$ mantissa digits:

$$x = (.123)_{10} \cdot 10^1, \qquad y = (.456)_{10} \cdot 10^{-1}, \qquad x+y = (.123 + .00456) \cdot 10^1 = (.12756)_{10} \cdot 10^1$$

For the code `z=x+y` the value `z` has to be a machine number. In this example this should be the machine number $z = fl\left(.12756 \cdot 10^1\right) = (.128)_{10} \cdot 10^1$.

Therefore the built-in machine operations like `y=sqrt(x)` are implemented as follows: For the given machine number x

- find the "exact" result $Y = \sqrt{x}$ (in practice: use some extra digits)
- return the machine number $y = fl(Y)$

Therefore machine operations don't return the exact result Y, but the closest possible machine number. Note that this causes an error $|\varepsilon_y| \le \varepsilon_M$. All built-in machine operations (like $x+y$, x/y, $\sqrt{x}$, $\sin x$) are implemented in this way. (Actually, for functions like $\sin(x)$ a slightly larger relative error $2\varepsilon_M$ is allowed to avoid the so-called "table-makers dilemma".)

Each arithmetic operation in a program causes a relative error of size $\le \varepsilon_M$.
Example: For the Matlab code `x=.1; y=1-cos(x)` the machine actually performs the following operations to find the computed value $\hat{y}$:

$\hat{x} := fl(.1)$	round .1 to closest machine number
$Y_1 := \cos(\hat{x})$	find true result $\cos(\hat{x})$ with extra accuracy
$\hat{y}_1 := fl(Y_1)$	round Y_1 to closest machine number
$Y := 1 - \hat{y}_1$	find true result $1 - \hat{y}_1$ with extra accuracy
$\hat{y} := fl(Y)$	round Y to closest machine number