MA2501 – Floating point systems

Brynjulf Owren

1 Floating point numbers

On a computer, the real numbers, $\mathbb{R}$, are approximated by a finite set of numbers, $\mathbb{F} \subset \mathbb{R}$. A way to represent a real number is to use a signed floating point format in the decimal number system

$$x = \pm\,(0.d_1 d_2 \cdots) \cdot 10^e \in \mathbb{R} \tag{1}$$

where each d_i is a decimal digit $0 \leq d_i \leq 9$, and where we can assume that $d_1 > 0$, and $e \in \mathbb{Z}$, the exponent. The representation is not unique, for instance

$$0.19999\cdots \text{ is the same as } 0.20000\cdots,$$

but that is not of any importance here. We could also replace the base 10 by any positive integer $b > 1$, and then the digits would be in the range $0 \leq d_i \leq b-1$. With this generalisation, we can write (1) in the form

$$x = \pm \sum_{k=1}^{\infty} d_k \cdot b^{e-k} \tag{2}$$

For the floating point numbers on a computer, we can only allow a finite number t of digits, $d_1, \ldots, d_t$. For any real number, we can introduce a map $\mathrm{fl} : \mathbb{R} \to \mathbb{F}$, by truncating or rounding the infinite number of digits in x, we would get

$$\mathrm{fl}(x) = \pm\,(0.d_1 d_2 \cdots d_t) \cdot b^e = \pm \sum_{k=1}^{t} d_k \cdot b^{e-k} \tag{3}$$

In addition to this, we also need to introduce a restriction on the set of exponents, typically $L \leq e \leq U$ for some integers L and U. Generally we can also write (3) (and even the real numbers) as

$$x = m \cdot b^e$$

where m is called the *mantissa*. Summarising, the characterising parameters of a floating point system are

- b The base of the number system, 10 in daily life, 2 or 16 on a computer.
- t The *precision*, the number of digits in the mantissa.
- L The smallest possible value of the exponent, e.
- U The largest possible value of the exponent, e.

2 Representation error

Before we start, we introduce some notation. We let the error be the difference between the exact value and the approximated value, and we put a bar over the symbol to signify the approximation value. Thus

$$\Delta_x = x - \bar{x}$$

is the absolute error Δ_x. We can also define the relative error as

$$\delta_x = \frac{x - \bar{x}}{\bar{x}}$$

Perhaps a better definition could have been to divide by x rather than by $\bar{x}$, but the chosen definition is often easier to work with, and we are usually in the regime where $|\delta_x| \ll |x|$.

Suppose now that $x \in \mathbb{R}$ is represented as $m_x \cdot b^e$ with $1/b \leq m_x < 1$. The floating point approximation to x is $\bar{x} = \mathrm{fl}(x)$ and the relative error is δ_x. We now assume that $\bar{x} = \bar{m}_x \cdot b^e$ where $\bar{m}_x$ is the full mantissa m_x chopped off after t digits. By comparing (2) with (3), ignoring the sign of x, we find

$$\delta_x = \frac{\sum_{k=t+1}^{\infty} d_k b^{-k}}{\sum_{k=1}^{t} d_k b^{-k}} \leq \frac{\sum_{k=t+1}^{\infty} (b-1) b^{-k}}{b^{-1}} = \frac{(b-1)b^{-(t+1)} \frac{1}{1-b^{-1}}}{b^{-1}} = b^{1-t}$$

It is interesting to observe that the bound for the representation error is independent of the size of the number x. One should keep in mind that this representation error analysis is only valid for values of x between the smallest and largest representable number.

2.1 Machine precision

An important quantity associated to a floating point system is the *machine precision*, also called *machine epsilon*. It is defined to be the smallest positive number η such that $\mathrm{fl}(1+\eta) > 1$. One can make a simple piece of Python code to compute it

```
error, const = float(1), float(2)
while const > 1.0:
    error = 0.5*error
    const = error + 1.

print(2*error)
```

where a typical output from the print could be: `2.220446049250313e-16`. But there is also a Python function that can be used to get hold of this number

```
import numpy as np
eta = np.finfo(float).eps
print(eta)
```

2.2 IEEE standard

Standards have been developed to have reasonable and consistent parameters of the floating point system, and IEEE is responsible for the most used standard. A floating point number in double precision occupies 64 bits. According to the standard, it allows for a precision $t = 53$ binary digits (bits) where one of them is implicitly given due to normalisation and so only 52 of the 64 digits are used for the mantissa. That leaves one sign digit, and 11 digits for the exponent, note that $2^{11} = 2048$. The exponent is required to be in the range $-1021 \leq e \leq 1024$ so 2046 patterns are used for the exponent. The remaining two patterns are used to signify `NaN` (not-a-number) and `Inf` (infinity). In python, one can import the module `sys` to get access to all the important parameters of the system.

```
import sys
sys.float_info
```

and the last statement causes the following printout

```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16,
radix=2, rounds=1)
```

There are similar standards for e.g. single precision (32 bits) floats.

2.3 Chopped and rounded arithmetic

Any $y \in \mathbb{R}$ can be written such that

$$|y| = m_y b^e + r_y b^{e-t}$$

where m_y is a t-digit mantissa and $0 \leq r_y \leq 1$. We assume that we have some information about r_y, this will be the case if there are one or more guard digits in the computational unit.

1. Chopped arithmetic, i.e. we ignore r_y and replace $|y|$ by $|\bar{y}| = m_y b^e$. We get the relative error

$$|\delta_y| = \left| \frac{r_y b^{e-t}}{m_y b^e} \right| \leq \frac{1 \cdot b^{e-t}}{\frac{1}{b} \cdot b^e} \leq b^{1-t}$$

2. Rounded arithmetic. Modify the mantissa as follows

$$|m'_y| = \begin{cases} |m_y| & \text{if } r_y < \frac{1}{2} \\ |m_y| + b^{-t} & \text{if } r_y \geq \frac{1}{2} \end{cases}$$

The resulting bound for the error is then

$$|\delta_y| \leq \frac{1}{2} \cdot b^{1-t}$$

3 The general law of error propagation

Two error sources contribute when we make a computation

1. There are errors in the input data to the computation, they may be enhanced or damped
2. The result of the computation must be converted to a floating point number in $\mathbb{F}$.

The first of these two error sources is usually approximated by linearisation. Suppose that we have n inputs $x_1, \ldots, x_n$, all potentially infected by an error, such that $x_k = \bar{x}_k + \Delta_k$. Ideally, we want to compute $y = \phi(x_1, \ldots, x_n)$ for some function $\phi : \mathbb{R}^n \to \mathbb{R}$, but in practice we can only obtain $\bar{y} = \phi(\bar{x}_1, \ldots, \bar{x}_n)$ and we set $\Delta_y = y - \bar{y}$. Assuming that ϕ is twice differentiable we find from Taylor expansion that

$$\bar{y} + \Delta_y = \phi(\bar{x}_1 + \Delta_1, \ldots, \bar{x}_n + \Delta_n) = \bar{y} + \sum_{i=1}^{n} \frac{\partial \phi}{\partial x_i}(\bar{x}_1, \ldots, \bar{x_n})\Delta_i + \mathcal{O}(\Delta^2)$$

We ignore the $\mathcal{O}(\Delta^2)$ terms and with a slight abuse of notation, we just set

$$\Delta_y = \sum_{i=1}^{n} \frac{\partial \bar{\phi}}{\partial x_i}\Delta_i$$

which is the general error propagation law. It is often more useful to consider relative errors, and by introducing

$$x_k = \bar{x}_k(1 + \delta_k),\ k = 1, \ldots, n, \quad y = \bar{y}(1 + \delta_y)$$

we get

$$|\delta_y| \leq \sum_{k=1}^{n} \left| \frac{\bar{x}_k}{\bar{\phi}} \right| \left| \frac{\partial \bar{\phi}}{\partial x_k} \right| |\delta_k|.$$

We can think of the quantities

$$\left| \frac{\bar{x}_k}{\bar{\phi}} \right| \left| \frac{\partial \bar{\phi}}{\partial x_k} \right|$$

as a *growth factor* for the input error source δ_k.

Example 3.1. *Addition. We let $\phi(x_1, x_2) = x_1 + x_2$, such that $\frac{\partial \phi}{\partial x_i} = 1,\ i = 1, 2$. Then*

$$|\delta_y| = |\delta_{x_1+x_2}| \leq \frac{|\bar{x_1}|}{|\bar{x}_1 + \bar{x}_2|}|\delta_1| + \frac{|\bar{x_2}|}{|\bar{x}_1 + \bar{x}_2|}|\delta_2|$$

Note in particular that the situation $\bar{x}_1 \approx -\bar{x}_2$ is problematic as the denominator becomes small and there can be an uncontrolled growth of error.

Example 3.2. *Multiplication. Now $\phi(x_1, x_2) = x_1 \cdot x_2$, such that $\frac{\partial \phi}{\partial x_1} = x_2$ and $\frac{\partial \phi}{\partial x_2} = x_1$. We find*

$$|\delta_y| = |\delta_{x_1 \cdot x_2}| \leq \frac{|\bar{x_1}|}{|\bar{x}_1 \cdot \bar{x}_2|} \cdot |\bar{x}_2| \cdot |\delta_1| + \frac{|\bar{x_2}|}{|\bar{x}_1 \cdot \bar{x}_2|} \cdot |\bar{x}_1| \cdot |\delta_2| = |\delta_1| + |\delta_2|$$

To summarise:

We use here a linear model for error propagation, which is an approximation to the true error. Evaluating a function ϕ on inaccurate input data $\bar{x}_1, \ldots, \bar{x}_n$, where $x_i = \bar{x}_i(1 + \delta_i)$ results in, after rounding the output to the nearest floating point number, in a relative error δ_y where

$$\delta_y = \sum_{i=1}^{n} \frac{\bar{x}_i}{\phi(\bar{x}_1, \ldots, \bar{x}_n)} \frac{\partial \phi}{\partial x_i}(\bar{x_1}, \ldots, \bar{x}_n)\, \delta_i + r, \quad |r| \leq \eta. \tag{4}$$

where η the machine epsilon of the system.