

**LANI TECHNOLOGY AND ICT SOLUTIONS**  
**SUMMARY**  
**COURSE CODE: CIT 335**  
**COURSE TITLE: Computational**  
**Science and Numerical**  
**Methods**

# **Real Numbers**

We begin with the number system commonly used in mathematical analysis and confront it with the more primitive number system available to us on any particular computer. We identify the basic constant (the machine precision) that determines the level of precision attainable on such computer.

One can introduce real numbers in many different ways. Mathematicians favor the axiomatic approach, which leads them to define the set of real numbers as a “complete Archimedean ordered field”. Here we adopt a more pedestrian attitude and consider the set of real numbers  $R$  to consist of positive and negative numbers represented in some appropriate number system and manipulated in the usual manner known from elementary arithmetic. We adopt here the binary number system, since it is the one most commonly used on computers.

# **Machine Numbers**

There are two kinds of machine numbers: floating point and fixed point. The first corresponds to the “scientific notation” in the decimal system, whereby a number is written as a decimal fraction times an integral power of 10. The second allows only for fractions. On a binary number, one consistently uses powers of 2 instead of 10. More important, the number of binary digits, both in the fraction and in the exponent of 2 (if any), is finite and cannot exceed certain limits that are characteristics of the particular computer at hand.

# Floating-Point Numbers

We denote by  $t$  the number of binary digits allowed by the computer in the fractional part and by  $s$  the number of binary digits in the exponent. Then the (real) floating-point numbers on that computer will be denoted by  $R(t, s)$ .

## Sources of Error

Basically there are three types of error that affect the result of a numerical computation

1. Errors in the input data are often unavoidable. The input data may be results of measurements with limited accuracy, or real numbers which must be represented with a fixed number of digits.
2. Rounding errors arise when computation are performed using a fixed number of digits.
3. Truncation errors arise when “an infinite process is replace by an infinite one”, e.g when an infinite series is approximated by a partial sum, or when a function is approximated by a straight line.

## Packaging of a fixed-point number in a machine register

Fixed-Point Numbers

This is the case (1.4) where  $e = 0$ . That is, fixed-point numbers are binary fractions,  $x = f$ , hence  $|f| < 1$ . We can therefore only deal with numbers that are in the interval  $(-1,1)$ . This, in particular, requires extensive scaling and

rescaling to make sure that all initial data, as well as all intermediate and final results, lie in that interval. Such a complication can only be justified in special circumstances where machine time and/or precision is at a premium. Note that on the same computer as considered before, we do not need to allocate space for the exponent in the machine register, and thus have in effect  $s + t$  binary digits available for the fraction  $f$

# Rounding

A machine register acts much like the infamous Procrustes bed in Greek mythology. Procrustes was the innkeeper whose inn had only beds of one size. If a fellow came along who was too tall to fit into his beds, he cut off his feet. If the fellow was too short, he stretched him. In the same way, if a real number comes along that is too long, its tail end (not the head) is cutoff; if it is too short, it is padded by zeros at the end.

# A Model of Machine Arithmetic

Any of the four basic arithmetic operations, when applied to two machine numbers, may produce a result no longer represented on the computer. We have therefore errors also associated with arithmetic operations. Barring the occurrence of overflow or underflow, we may assume as a model of machine arithmetic that each arithmetic operation  $o$  ( $= +, -, \times, /$ ) produces a correctly rounded result. Thus, if  $x, y \in R(t, s)$  are floating-point machine numbers, and  $fl(xoy)$  denotes the machine-produced result of the arithmetic operation  $xoy$ .

# Error Propagation in Arithmetic Operations: Cancellation Error

when we are dealing with machine operations.) Our interest is in the errors in the results caused by errors in the data.

a) Multiplication We consider values  $x(1 + \epsilon_x)$  and  $y(1 + \epsilon_y)$  of  $x$  and  $y$  contaminated by relative errors  $\epsilon_x$  and  $\epsilon_y$ , respectively. What is the relative error in the product? We assume  $\epsilon_x, \epsilon_y$  sufficiently

small so that quantities of second order,  $\epsilon^2_x, \epsilon_y \epsilon_x$  and  $\epsilon^2_y$  – and even more so, quantities of still higher order – can be neglected against the epsilons themselves. Then

$$x(1 + \epsilon_x) \cdot y(1 + \epsilon_y) = x \cdot y(1 + \epsilon_x + \epsilon_y + \epsilon_x \epsilon_y) \approx x \cdot y(1 + \epsilon_x + \epsilon_y).$$

Thus, the relative error  $\epsilon_{x \cdot y}$  in the product is given (at least approximately) by  $\epsilon_{x \cdot y} = \epsilon_x + \epsilon_y$ , (1.16)

that is, the (relative) errors in the data are being added to produce the (relative) error in the result. We consider this to be acceptable error propagation, and in this sense, multiplication is a benign operation.

b) Division. Here we have similarly (if  $y \neq 0$ )

$$x(1 + \epsilon_x) / y(1 + \epsilon_y) = x(1 + \epsilon_x)(1 - \epsilon_y + \epsilon^2_y - \dots)$$

that is,

$$y(1 + \epsilon_y) \cdot y$$

$$\approx x(1 + \epsilon_x - \epsilon_y), \quad y$$

$$\epsilon_x = \epsilon_x - \epsilon_y. \quad y$$

Also, division is a benign operation.

c) Addition and subtraction. Since  $x$  and  $y$  can be numbers of arbitrary signs, it suffices to look at addition. We have

## The Condition of a Problem

$y =$

A problem typically has an input and an output. The input consists of a set of data, say, the coefficients of some equation, and the output of another set of numbers uniquely determined by the input, say, all the roots of the equation in some prescribed order.

## The Condition of a Problem

A problem typically has an input and an output. The input consists of a set of data, say, the coefficients of some equation, and the output of another set of numbers uniquely determined by the input, say, all the roots of the equation in some prescribed order.

Error propagation, a term that refers to the way in which, at a given stage of a calculation, part of the error arises out of the error at a previous stage. This is

independent of the further round off inevitably introduced between the two stages. Unfavorable error propagation can seriously affect the results of a calculation. The investigation of error propagation in simple arithmetical operations is used as the basis for the detailed analysis of more extensive calculations.

## ROUNDING Errors in Floating Point Representation

When number are represented in the floating point system  $(\beta, t, L, U)$ , we get rounding errors because of the limiting precision. We shall derive a bound for the relative error.

Assume that a real number  $X \neq 0$  can be written (exactly)

$$X = M \cdot \beta^e, 1 \leq |M| < \beta,$$

And let  $x = m \cdot \beta^e$ , where  $m$  is equal to  $M$ , rounded to  $t + 1$  digits. Then

$$|m - M| \leq \frac{1}{2} \beta^{-t},$$

And we get a bound for the absolute error,  $|x - X| \leq \frac{1}{2} \beta^{-t} \cdot \beta^e$ .

2

This leads to the following bound for the relative error:

$$\frac{|x - X|}{|X|} \leq \frac{\frac{1}{2} \beta^{-t} \cdot \beta^e}{|M| \cdot \beta^e} = \frac{1}{2} \beta^{-t} \frac{1}{|M|} \leq \frac{1}{2} \beta^{-t} \frac{1}{1} = \frac{1}{2} \beta^{-t}$$

$$\frac{|x - X|}{|X|} \leq \frac{1}{2} \beta^{-t} \frac{1}{|M|} \leq \frac{1}{2} \beta^{-t}$$

The last inequality follows from the condition  $|M| \geq 1$ .

## FLOATING POINT ADDITION

```

z = x + y;
ez := ex;
if ex - ey ≥ t + 3 then
  mz := mx;
else
  endif
(ex ≥ ey is assumed)
(right shift ex - ey positions)
(see below)
my := my / βex-ey; mz := mx + my; Normalize;
if ex - ey < t + 3, then my can be stored exactly after the shift, since the
arithmetic register is assumed to hold 2t + 4 digits. Also the addition mx +

```

my is performed without error. In general, the result of these operations may be an unnormalized floating point number  $z = m_z \cdot \beta^{e_z}$ , with  $[m_z] \geq \beta$  or  $[m_z] < 1$ , e.g.  $5.67 \cdot 100 = 10.245 \cdot 100$  or  $5.678 \cdot 100 + (-5.612 \cdot 100) = 0.066 \cdot 100$ . In such cases the floating point number is normalized by appropriate shifts. Further, the significand must be rounded to  $t+1$  digits. These two tasks are performed by the following algorithm that takes an unnormalized, nonzero floating point number  $m \cdot \beta^e$  as input and gives a normalized floating number  $x$  as output.

NORMALIZE if  $|m| \geq \beta$  then

$m := m/\beta$ ;  $e := e + 1$ ;

else

While  $|m| < 1$  do

$m := m \cdot \beta$ ;  $e := e - 1$ ;

(right shift one position)

(left shift one position)

endif

Round  $m$  to  $t+1$  digits; If  $|m| = \beta$  then

$m := m/\beta$ ;  $e := e + 1$ ;

endif

if  $e > U$  then

$x := \text{Inf}$ ; elseif  $e < L$  then

$x := 0$ ;

$x = m \cdot \beta^e$ ;

(right shift one position)

(exponent overflow)

(exponent underflow)

(the normal class)

else

endif

The if-statement after the rounding is needed because the rounding to  $t + 1$  digits can give an unnormalized results:

$9.9995 \cdot 10^3 = 10.000 \cdot 10^3 = 1.000 \cdot 10^4$ . The multiplication and division algorithms are simple:

# FLOATING POINT MULTIPLICATION

$z := x * y$ ;

$e_z := e_x + e_y$ ;

$m_z = m_x * m_y$

Normalize:

## FLOATING POINT DIVISION

```
z := x / y;  
if y = 0 then  
  division by zero; (error signal5) else  
  ez := ex - ey;  
  
mz = mx / my Normlize; endif
```

## IEEE Standards documents (standards,

recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association ("IEEE-SA") Standards Board. IEEE ("the Institute") develops its standards through a consensus development process, approved by the American National Standards Institute ("ANSI"), which brings together volunteers representing varied viewpoints and interests to achieve the final product.

## Approximation and Interpolation

The present chapter is basically concerned with the approximation of functions. The functions in question may be functions defined only on a finite set of points. The first instance arises, for example, in the context of special functions (elementary or transcendental) that one wishes to evaluate as part of a subroutine. Since any such evaluation must be reduced to a finite number of arithmetic operations, we must ultimately approximate the function by means of a polynomial or a rational function. The second instance is

frequently encountered in the physical sciences when measurements are taken of a certain physical quantity as a function of some other physical quantity (such as time). In either case one wants to approximate the given function “as well as possible” in terms of other simpler functions.

## Inner Products

Given a continuous or discrete measure  $d\lambda$ , as introduced earlier, and given any two functions  $u, v$  having a finite norm (2.8), we can define the inner product

$$(u, v) = \int u(t) v(t) d\lambda(t) \quad (2.10) \quad R$$

(Schwarz’s inequality  $| (u, v) | \leq \|u\|_{2, d\lambda} \cdot \|v\|_{2, d\lambda}$ , cf. Ex. 6, tells us that the integral in (2.10) is well defined.)

The inner product (2.10) has the following obvious (but useful) properties:

1. Symmetry:  $(u, v) = (v, u)$ ;
2. Homogeneity:  $(\alpha u, v) = \alpha (u, v)$ ,  $\alpha \in \mathbb{R}$ ;
3. Additivity:  $(u + v, w) = (u, w) + (v, w)$ ; and
4. Positive definiteness:  $(u, u) \geq 0$ , with equality holding if and only if  $u \equiv 0$  on  $\text{supp } d\lambda$ .
5. Homogeneity and Additivity together give linearity,

Then each  $\pi_j$  so determined is orthogonal to all preceding ones.

## Convergence

We have seen in Sect. 2.1.2 that if the class  $\Phi = \{\phi_n\}$  consists of  $n$  functions  $\pi_j$ ,  $j = 1, 2, \dots, n$ , that are linearly independent on the support of some measure  $d\lambda$ , then the least squares problem for this measure.