Numerical Computing: An Introduction

Gyula Horváth

Horvath@inf.u-szeged.hu

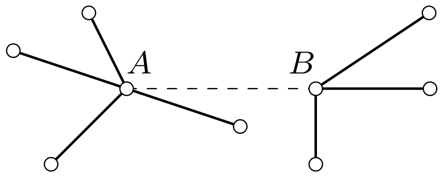
University of Szeged
Hungary

Tom Verhoeff

T.Verhoeff@TUE.NL

Eindhoven University of Technology
The Netherlands

IOI 2002: Bus Terminals



Given a set of points with integer coordinates,

select two points as ‘hubs’ and

assign each of the remaining points to a hub,

while minimizing the maximum value (over all P, Q) of

$$c(P, Q) \; = \; d(P, H(P)) + d(H(P), H(Q)) + d(H(Q), Q)$$

Integer Computations

$\mathbb{Z}$ = the set of **integers**

How well is **integer arithmetic** implemented on a computer?

Non-Integer Numbers

- Fractions, percentages, fixed-point currency values
- Real numbers, complex numbers
- Scientific notation: 6.022142×10^{23}
- Floating-point types in programming languages

How well is **non-integer arithmetic** implemented on a computer?

Quote from Donald E. Knuth

"Floating point computation is by nature inexact, and programmers can easily misuse it so that the computed answers consist almost entirely of "noise." One of the principal problems of numerical analysis is to determine how accurate the results of certain numerical methods will be. There is a "credibility-gap": We don't know how much of the computer's answers to believe. Novice computer users solve this problem by implicitly trusting in the computer as an infallible authority; they tend to believe that all digits of a printed answer are significant. Disillusioned computer users have just the opposite approach; they are constantly afraid that their answers are almost meaningless."

The Art of Computer Programming, Vol. 2: Seminumerical Algorithms (3rd Ed.), Addison-Wesley, 1998, §4.2.2.

Count Down

Pascal	C

`const D = 0.1;`
`var x: Real;`
 	
`begin`
`  x := 1.0`
`;`
`  while x > 0.0 do`
`    x := x - D`
`;`
`  writeln ( x:1:2 )`
`end.`
	`#include <stdio.h>`
	`#define D    0.1`
	
	`int main ( void )`
	`{ double x = 1.0;`
	
	`  while ( x > 0.0 )`
	`    x = x - D;`
	
	`  printf ( "%1.2f\n", x );`
	`}`

What value does this program print?

Euclidean Paths

A	B	C	D
$(2,5,31)$	$(1,2,9)$	$(0,7,27)$	$(1,8,10)$

Consider the two V-shaped paths via the origin O: AOB and COD.

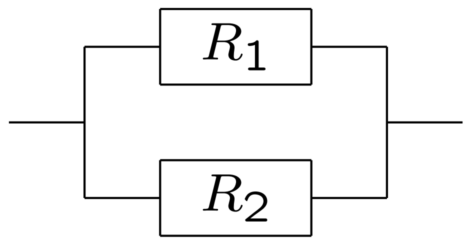
Are the lengths of these two paths equal?

If not, which is bigger?

Now also tackle the case with

A	B	C	D
$(4,12,28)$	$(2,6,14)$	$(1,1,23)$	$(1,13,19)$

Parallel Resistors



Write a program to compute the effective resistance,

given the non-negative values R_1 and R_2 as input.

Quadratic Equation

Consider the equation

$$ax^2 + bx + c \;=\; 0 \tag{1}$$

where parameters a, b, and c are given real constants and x is a real variable, whose value(s) satisfying (1) must be determined.

What conditions to impose on the parameters to make this into a reasonable programming assignment?

Solve your assignment.

How to determine the quality of solver programs?

Floating-Point Numbers

$\mathbb{R}$ = the set of **real numbers**

Consider integers $\beta \geq 2, \quad t \geq 1, \quad e_{\mathsf{min}} \leq e_{\mathsf{max}}$

$\mathbb{F}(\beta, t, e_{\mathsf{min}}, e_{\mathsf{max}})$ = the set of **floating-point numbers** x of the form

$$x \;=\; \pm f \times \beta^e$$

where **fraction** f and **exponent** e satisfy:

- $f \times \beta^t$ is an integer with $f = 0$ or $1 \leq |f| < \beta$, and

- e is an integer with $e_{\mathsf{min}} \leq e \leq e_{\mathsf{max}}$

Floating-Point Parameters

β is called the **base** of $\mathbb{F}$; typically $\beta = 2$

$p = t + 1$ = the number of bits in the binary representation of f;
p is called the **precision** of $\mathbb{F}$

The smallest $\mathbb{F}$-number larger than 1 is $1 + \epsilon$ with $\epsilon = \beta^{-t}$;
ϵ is called the **machine epsilon** of $\mathbb{F}$.

The interval from the smallest positive $\mathbb{F}$-number $N_{\text{min}} = \beta^{e_{\text{min}}}$ to the largest one $N_{\text{max}} = (\beta - \epsilon)\beta^{e_{\text{max}}}$ is called the **range** of $\mathbb{F}$.

IEEE Standard: Normalized Binary Floating-Point Numbers

Type	Parameter values					
	β	t	e_{min}	e_{max}	ϵ	Range
Single	2	23	-126	127	$2^{-23} \approx 1.2 \times 10^{-7}$	$\approx 10^{\pm 38}$
Double	2	52	-1022	1023	$2^{-52} \approx 2.2 \times 10^{-16}$	$\approx 10^{\pm 308}$

Type	Sizes in bits			
	$\pm$	f	e	Total
Single	1	23	8	32
Double	1	52	11	64

Floating-Point Operations

Most operations on $\mathbb{R}$ are not closed in $\mathbb{F}$.

When such operations are simulated on a computer, the result is forced into $\mathbb{F}$, yielding an **approximation** of the exact result.

This introduces a (small) **rounding error** into floating-point calculations. Subsequent operations on inexact results can magnify, or reduce, the error in non-intuitive ways.

The aim of **error analysis** is to understand the propagation of errors in numerical algorithms, in particular to prove bounds on the error in the final result.

Floating-Point Arithmetic

Approximation function $fl : \mathbb{R} \longrightarrow \mathbb{F}$

$fl(x)$ is the floating-point number nearest to real number x

For operation $\diamond$ on $\mathbb{R}$, let $\widehat{\diamond}$ be its implementation on $\mathbb{F}$

IEEE Standard requires 'best' results:

$$x \mathbin{\widehat{\diamond}} y \quad = \quad fl(x \diamond y)$$

for all $\diamond \in \{+, -, \times, /\}$ and $x, y \in \mathbb{F}$

Floating-Point Arithmetic: Limitations

To what extent is $\mathbb{F}$ an adequate model of $\mathbb{R}$?

Which mathematical laws hold when translated from $\mathbb{R}$ to $\mathbb{F}$?

$$\begin{array}{ccc} \mathbb{R}^n & \xrightarrow{fl^n} & \mathbb{F}^n \\ \downarrow\mathcal{A} & & \downarrow\widehat{\mathcal{A}} \\ \mathbb{R} & \xrightarrow{fl} & \mathbb{F} \end{array}$$

For all $\diamond \in \{+, -, \times, /\}$ and $x, y \in \mathbb{R}$

$$fl(x \diamond y) \;=\; fl(x) \,\widehat{\diamond}\, fl(y)$$

Floating-Point Arithmetic: Examples

Consider a machine working with two decimal digits $(\beta, t = 10, 1)$

$$fl(1.06 + 3.06) \;=\; fl(4.12) \;=\; 4.1$$

$$fl(1.06) \mathbin{\widehat{+}} fl(3.06) \;=\; 1.1 \mathbin{\widehat{+}} 3.1 \;=\; 4.2$$

How do the following expressions compare:

$$5.3 \times 0.2 + 5.1 \times 0.6 \quad ? \quad 1.1 \times 1.9 + 5.1 \times 0.4$$

Exact evaluation yields:

$$1.06 + 3.06 \;<\; 2.09 + 2.04$$

Machine approximation yields:

$$1.1 + 3.1 \;>\; 2.1 + 2.0$$

Count Down: Analysis

$D = 0.1$ has infinite repeating binary representation:

$$(0.0001100110011001100110011001100\ldots)_2 \;=\; \sum_{k=1}^{\infty} 3/2^{4k+1}$$

Cannot be represented exactly as a binary floating-point number

In the program $D = fl(0.1) \neq 0.1$

Double versus Single

0.1 versus 0.01

Euclidean Paths: Analysis

Pythagoras' Theorem yields:

$$
\begin{aligned}
AOB &= \sqrt{990} + \sqrt{86} \approx 40.73788394060\ldots \\
COD &= \sqrt{778} + \sqrt{165} \approx 40.73788394062\ldots
\end{aligned}
$$

The two lengths coincide on the 12 most significant decimal digits, with a difference on the order of 10^{-11}.

For the second pair we find

$$
\begin{aligned}
AOB &= \sqrt{944} + \sqrt{236} \approx 46.086874487211645\ldots \\
COD &= \sqrt{531} + \sqrt{531} \approx 46.086874487211652\ldots
\end{aligned}
$$

where the difference is less than 10^{-14}.

Are the lengths really different?

Euclidean Paths: Analysis

For the second pair, factorization leads to a confirmation:

$$\begin{aligned}\sqrt{944}+\sqrt{236} &= \sqrt{16\cdot 59}+\sqrt{4\cdot 59} = 6\sqrt{59}\\ \sqrt{531}+\sqrt{531} &= \sqrt{9\cdot 59}+\sqrt{9\cdot 59} = 6\sqrt{59}\end{aligned}$$

For the first pair, three squarings lead to a contradiction:

$$\begin{aligned}\sqrt{990}+\sqrt{86} &= \sqrt{778}+\sqrt{165}\\ 990+2\sqrt{990\cdot 86}+86 &= 778+2\sqrt{778\cdot 165}+165\\ 133 &= 2\cdot\left(\sqrt{778\cdot 165}-\sqrt{990\cdot 86}\right)\\ 133^2 &= 4\cdot\left(778\cdot 165-2\sqrt{778\cdot 165\cdot 990\cdot 86}+990\cdot 86\right)\\ 8\sqrt{778\cdot 165\cdot 990\cdot 86} &= 4\cdot(778\cdot 165+990\cdot 86)-133^2\\ 64\cdot 778\cdot 165\cdot 990\cdot 86 &= 836351^2\\ 699482995200 &= 699482995201\end{aligned}$$

Parallel Resistors: Analysis

Replacement resistance R for two parallel resistors R_1 and R_2:

$$R = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}} = \frac{R_1 \cdot R_2}{R_1 + R_2}$$

What if $R_1 = 0$ and/or $R_2 = 0$?

IEEE Standard supports well-behaved **infinities**:

$$1/0 = \infty \qquad \infty + x = \infty \qquad 1/\infty = 0$$

However, $0/0$ is undefined, yielding a **NaN** (not-a-number)

Quadratic Equation: Analysis

The well-knownn a, b, c-formula for solving quadratic equations:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

Applying it to

$$10^{-8} \times x^2 + x - 1 = 0 \tag{3}$$

and evaluating it in IEEE single precision, yields

$$x_{1,2} = 0.000000000, \quad -1.000000000 \times 10^8$$

Should have been

$$x_{1,2} = \boxed{1}.000000000, \quad -1.000000000 \times 10^8$$

Quadratic Equation: Analysis

For our positive root, $-b$ and $+\sqrt{b^2 - 4ac}$ have opposite signs and are of almost equal magnitude, because $|4ac| \ll b^2$.

When adding them, the (roundoff) error present in the computed value for $b^2 - 4ac$ is suddenly magnified enormously in relative size. This phenomena is known as **cancellation**.

Cancellation is avoided in the less-known alternative formula:

$$x_{1,2} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}} \tag{4}$$

Measures for Accuracy

Suppose the exact value $x \in \mathbb{R}$ is approximated by $\widehat{x} \in \mathbb{F}$.

The **absolute error** (in $\widehat{x}$ for x) is defined as

$$|x - \widehat{x}|$$

The **relative error** is defined as

$$\frac{|x - \widehat{x}|}{|x|}$$

Scientific and engineering applications often involve scaling, e.g. when converting values to other units.

The relative error is preferred because it is **invariant under scaling**.

Stability of Numerical Algorithm

A numerical algorithm is called **stable**, when it produces answers whose accuracy is on the order of what can 'reasonably' be expected for the problem at hand.

Challenges in numerical mathematics are

- to determine what can 'reasonably' be expected and
- to construct appropriate stable algorithms.

For the positive root of (3), the a, b, c-formula (2) is unstable, whereas the alternative formula (4) is stable.

Quadratic Equation: Further Analysis

Cancellation is also possible in the subtraction $b^2 - 4ac$ when $b^2 \approx 4ac$.

In this case it is harder to circumvent, because it is inherent in the problem itself and not a consequence of a badly chosen algorithm. Determining the roots when they are nearly equal is said to be an **ill-conditioned problem**.

The squaring b^2, the multiplication $4ac$, and the final division by $2a$ can produce (intermediate) results that fall outside the representable range. This is referred to as **underflow** or **overflow**.

For b^2 and $4ac$ this can happen even if the final results are representable within the range of floating-point numbers.

Quadratic Equation: Complications

1. Restrictions on the input coefficients a, b, c

2. Roots that are not representable within the floating-point range

3. Complex roots

4. Desired accuracy of the output roots

5. Evaluation of a quadratic-solving program

Error Analysis

Estimate quantitatively the error in a computation: e.g. give bounds

Given floating-point numbers A, B, X, compute $Y = AX + B$.

What can be said about the error in $\widehat{Y} = A \mathbin{\widehat{\times}} X \mathbin{\widehat{+}} B$?

$$\begin{aligned}
F(A, B, X) &= AX + B \\[1em]
\widehat{F}(A, B, X) &= A \mathbin{\widehat{\times}} X \mathbin{\widehat{+}} B \\
&= AX(1+\delta) \mathbin{\widehat{+}} B \\
&= (AX(1+\delta) + B)\,(1+\eta)
\end{aligned}$$

with $|\delta|, |\eta| \leq \epsilon/2$

Forward Error Analysis

$$\begin{aligned}\widehat{F}(A,B,X) &= (AX+B)(1+\eta) + AX\delta(1+\eta) \\ &= F(A,B,X) + (AX+B)\eta + AX\delta(1+\eta)\end{aligned}$$

$\widehat{F}$ computes exact value plus a perturbation (forward error):

$$(AX+B)\eta + AX\delta(1+\eta)$$

- Absolute error $\approx AX(\delta+\eta) + B\eta$: no reasonable bound
- Relative error $\approx \frac{AX}{AX+B}\delta + \eta$: no reasonable bound
- Error always small compared to B: false
- Error always small compared to AX: false

Error Analysis Is Not Easy

- Error propagation is a complex process

- Statistical analysis is not applicable if there are just a few steps

  It is not reliable (if there are many steps: law of large numbers), because errors need not be independent but can be correlated; in that case, statistical analysis is too optimistic

- Interval arithmetic often is (far) too pessimistic; errors can and often do (partially) cancel each other

Backward Error Analysis

$$
\begin{aligned}
\widehat{F}(A,B,X) &= (AX(1+\delta)+B)\,(1+\eta) \\
&= A(1+\eta)X(1+\delta)+B(1+\eta) \\
&= F(A(1+\eta),B(1+\eta),X(1+\delta)) \\
&= \boxed{F(\hat{A},\hat{B},\widehat{X})}
\end{aligned}
$$

where

$$
\begin{aligned}
\hat{A} &= A(1+\eta) \\
\hat{B} &= B(1+\eta) \\
\widehat{X} &= X(1+\delta)
\end{aligned}
$$

$\widehat{F}$ computes exact solution for slightly perturbed input.

Compare this error to the error already present in A, B, X.

Other Areas in Numerical Mathematics

Two additional sources of error:

Data Uncertainty: the error already present in the input values

E.g. by physical measurement

Truncation Error: the error introduced by an inexact algorithm, which is known to produce incorrect answers when run on an ideal machine, with the purpose of obtaining accurate answers in less time

E.g. by chopping off an infinite series or approximating a function by a polynomial.

Recommendation 1

Avoid floating-point numbers in computing whenever possible.

To teachers: When designing programming problems, there are plenty of possibilities without floating-point numbers.

In fact, it is a good attitude to *forbid* your students to use floating-point numbers in their programs, because it is so hard to reason about floating-point programs.

To students: Resist the temptation to use floating-point numbers when solving programming problems whose specification does not involve them.

Recommendation 2

If you do want to use floating-point numbers, study the literature.

To teachers: When setting a programming problem involving floating-point numbers, the constraints must be expressed carefully and the problem must be solvable for all allowed inputs. Avoid ill-conditioned problems.

To students: Before resorting to floating-point numbers, convince yourself that this is really necessary.

Then, convince yourself that your program satisfies all constraints. In particular, check that you have not fallen into one of the 'standard' traps giving rise to an unstable algorithm.

In both cases, some form of *error analysis* is needed.

Quote from Donald E. Knuth (continued)

“Many serious mathematicians have attempted to analyze a sequence of floating point operations rigorously, but have found the task so formidable that they have tried to be content with plausibility arguments instead.”