What can you learn about

Floating-Point Arithmetic

in One Hour ?

. . . .

by Prof. W. Kahan
Univ. of Calif. @ Berkeley

prepared for CS 267,
(Profs. J.W. Demmel of UCB & A. Edelman of MIT)
8 Feb. 1996

Numbers in Computers:

(Character Strings ... get Converted to or from ...)

Integers

Fixed-Point

Floating-Point

Integers

..., -3, -2, -1, 0, 1, 2, 3, ...

In all programming languages.

+, -, x are *Exact* unless they Overflow.

Overflow thresholds determined by
(un)signed
Radix (2 or 10)
wordsize (1 byte, 2 bytes, 4 bytes, 8 bytes, ...)
(cf. *type*).

Division ==> Quotient and Remainder.

Fixed-Point

-0.712 , 1.539 , 27.962 , 745.288 , ...

Provided directly in COBOL , ADA ; otherwise simulated.

+ , - , x by Integer are exact unless they Overflow
x , / *Rounded Off* to a fixed number of digits after the point.

{ Available numbers } = { integers } / (Scale Factor) ;
Scale Factor = Power of 2 or 10 ,
selected by programmer to determine a
format or *type* .

Floating-Point

-7.12 E-01 , 1.539 E 00 , 2.7962 E 01 , 7.45288 E 02 , ...
(cf. "Scientific Notation")

Called REAL, float, DOUBLE PRECISION, ...

Every arithmetic operation is rounded off to fit a
Destination Format or *Type* depending upon
language conventions and
computer register-architecture (... Compiler).

Too Big for destination ==> Overflow.
Nonzero but Too Tiny ==> Underflow.

(Despite rounding, some operations are Exact ; e.g., X := -Y .)

Logarithmic Floating-Point

$\{\text{Available values}\} = \pm(10 \text{ or } 2)^{\{\text{Fixed Point numbers}\}}$

Absent Over/Underflow, x and / are Exact , and
Distributive Law $X\cdot(Y+Z) = X\cdot Y + X\cdot Z$ persists.

But

Subtract is difficult to implement to near-full precision.
Add, subtract are slow unless precision is short, < 6 sig. dec.
Can't represent small integers 2 and 3 exactly.

Used only in a few embedded systems.

Conventional Floating-Point

{ Available values } = {long integers}·Radix$^{\{\text{short integers}\}}$

Radix = 2 or 10 or 16 .

Some also have ∞ , NaN / Indefinite / Reserved Operand.

Models of Roundoff

Let operation • come from { + , - , x , / } ; then,

absent Over/Underflow,

Computed[X•Y] = (X•Y)·(1 + ß) for some tiny ß ;

| ß | < Radix$^{(- \#\text{Sig. Digits})}$ roughly ,

except for CRAY X-MP, Y-MP, C90, J90

which have peculiar arithmetic.

CRAY X-MP, Y-MP, C90, J90 have peculiar arithmetic.

e.g.: $1 \cdot X$ can Overflow if $|X|$ is big enough, $\approx 10^{2466}$

Abbreviated multiply, composite divide:
$X/Y \longrightarrow \approx X \cdot (1/Y)$.

Consequently, absent Over/Underflow or 0/0 ,

$-1 \leq X/\sqrt{(X^2 + Y^2)} \leq 1$ despite 5 rounding errors

on all H-P calculators since 1976 and on EVERY commercially significant computer EXCEPT a CRAY.
(Proof of inequality easy only with IEEE 754.)

CRAYs Lack GUARD DIGIT for Subtraction:

Pretend 4 sig. dec.; compute 1.000 - 0.9999 :

With guard digit:

```
  1.000
- 0.9999
-----------------------
  0.0001    —>                    1.000 · 10^-4
```

Without guard digit

```
  1.000     —>    1.000
- 0.9999    —>  - 0.999
---------------  ---------------
                  0.001   —>  1.000 · 10^-3
```

Violates Theorem: If P and Q are floating-point numbers in the same format, and if $1/2 \le P/Q \le 2$, then P - Q is computable Exactly unless it Underflows (which it can't in IEEE 754).

Programs that can FAIL only on a CRAY

for lack of a guard bit:

Computations with Divided Differences.

Area and Angles of a Triangle, given its side-lengths.

Roundoff suppression in solutions of Initial-Value Problems.

Software simulations of Doubled-Double precision.

Divide-and-Conquer Symmetric Eigenproblems (Ming Gu’s)

cured in LAPACK by performing operation X := (X+X) - X
to shear off X’s last digit only on CRAYs (and hex. IBM 3090).

Why is CRAY's arithmetic so Aberrant ?

Aberration "justified" by misapplication of principles behind ...

Backward Error Analysis: The computed value F(X) of a desired function f(X) is often acceptable if F(X) = f(X') for some (unknown) X' practically indistinguishable from X . For example, the solution f of the linear system X·f = y is often considered adequately approximated by F satisfying X'·F = y' even if, when X is nearly singular, F is utterly different from f , since the residual y - X·F is still very tiny.

e.g.: Subtraction X - Y without a Guard Digit is no worse than replacing X by X' and Y by Y' . For instance, in the example with 4 sig. dec., X = 1.000 and Y = 0.9999 are replaced by Y' = Y and X' = 1.0009 with an error smaller than 1 ulp.

The foregoing "justification" for omitting a guard digit ignores

Correlations:

Example:

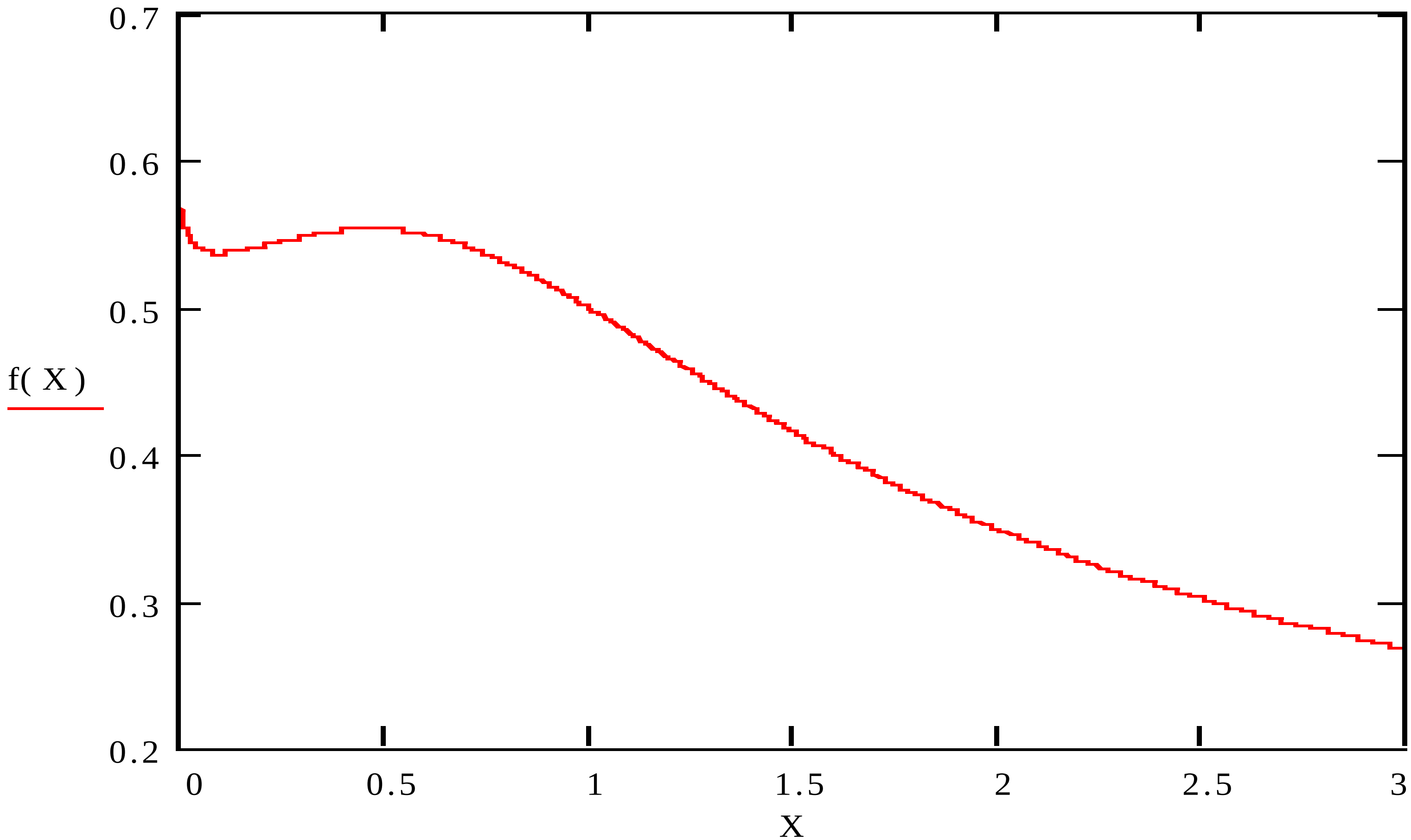
Real Function f(Real x) :=
if x < 0 then Shout " Invalid f(Negative) ."
else if x = 1 then 0.5
else if x < 1 then $-\arctan(\ln(x))/\arccos(x)^2$
else $\arctan(\ln(x))/\operatorname{arccosh}(x)^2$.

This f(x) is a smooth function despite the branch; if $|x-1| < 1$,

$$f(x) = 1/2 - (x-1)/6 + (x-1)^2/20 + 124(x-1)^3/945 + \ldots .$$

$$f(x) := \text{if}\left(x=1,\ 0.5,\ \text{if}\left(x<1,\ \frac{-\text{atan}(\ln(x))}{\text{acos}(x)^2},\ \frac{\text{atan}(\ln(x))}{\text{acosh}(x)^2}\right)\right)$$

$$X := 0.0000001,\ 0.001\,..\,3$$



If you believe computers may replace ln(x) by ln(x’) , and either arccos(x) by arccos(x”) or arccosh(x) by arccosh(x”) , where x’ and x” are uncorrelated but differ from x by at most 1 ulp., then you must infer that expressions

$-\arctan(\ln(x'))/\arccos(x'')^2$ and $\arctan(\ln(x'))/\mathrm{arccosh}(x'')^2$

become unreliable like 0.0/0.0 as x —> 1.0 , so you must modify the program; choose some small threshold T > 0 and ...

Real Function f(Real x) :=

if x < 0 then Shout “ Invalid f(Negative) .”

else if |x-1| < T then $1/2 - (x-1)/6 + (x-1)^2/20 + 124(x-1)^3/945$

else if x < 1 then $-\arctan(\ln(x))/\arccos(x)^2$

else $\arctan(\ln(x))/\mathrm{arccosh}(x)^2$.

This modification actually LOSES accuracy, even on a CRAY !

Characterizations of Floating-Point Arithmetic

Prescriptive:

Computer's Assembly-language manuals or
Circuit diagram Too diverse !

Descriptive:

Axioms like X•Y —> (X•Y)·(1 + ß) & | ß | < ... (CRAY)
X+Y = Y+X , X·Y = Y·X (CRAY)
X-Y = -(Y-X) (GE / Honeywell)
Monotonicity ... (CRAY)

No tractable set of axioms that covers all commercially significant computers and H-P calculators of the past decade suffices to prove

$$-1 \le X/\sqrt{X^2 + Y^2} \le 1$$ despite 5 rounding errors

IEEE Standard 754 for Binary Floating-Point Arithmetic Prescribes

Algebraic Operations

+ - * / √ remainder compare

Conversions

Decimal <—> Binary

Integer <—> Single <—> Double <—> ...

upon and among a small number of Floating-Point Formats, each with its own ...

NaNs (Not-a-Number),

±∞ (Infinity), and

Finite real numbers all of the simple form $2^{k+1-N}\,n$:

integer n (signed *Significand*),

integer k (unbiased signed *Exponent*);

Finite real numbers all of the simple form $2^{k+1-N} n$:

K+1 Exponent bits: $1 - 2^K < k < 2^K$, and
N Significant bits: $-2^N < n < 2^N$.

Table of Formats' Names & Parameters:

Status	IEEE 754 Format	Fortran	C	Bytes	K+1	N
Obligatory	Single	REAL*4	float	4	8	24
Ubiquitous	Double	REAL*8	double	8	11	53
Optional Intel, M680x0	Double-Extended	REAL*10 REAL*12	long double	≥ 10	≥ 15	≥ 64
Unimplemented SPARC / SGI / HP	Quadruple (by Consensus)	REAL*16	long double	16	15	113
Software POWER-PC	*Doubled-Double* *NOT standard*	*REAL*16*	*long* *double*	*16*	*11*	*≈105*

cf. notes p. 2

Names of Floating-Point Formats:

Single-Precision	`float`	REAL*4
Double-Precision	`double`	REAL*8
Double-**Extended**	`long double`	REAL*10 or 12 ... Intel, Motorola
Doubled-Double	`long double`	REAL*16 in software.
Quadruple-Precision	`long double`	REAL*16

(Except for the IBM 3090, no current computer supports either of the last two formats fully in its hardware; at best they are simulated in software too slowly to run routinely, so we disregard them.)

Spans and Precisions of Floating-Point Formats :

Format	Min. Normal	Max. Finite	Rel. Prec'n	Sig. Dec.
IEEE Single	1.2 E-38	3.4 E 38	5.96 E-8	6 - 9
IEEE Double	2.2 E-308	1.8 E 308	1.11 E-16	15 - 17
IEEE Double Extended	3.4 E-4932	1.2 E 4932	5.42 E-20	18 - 21
Doubled-Double	2.2 E-308	1.8 E 308	≈ 1.0 E-32	≈ 32
Quadruple	3.4 E-4932	1.2 E 4932	9.63 E-35	33 - 36
IBM hex. REAL*4	5.4 E-79	7.2 E 75	9.5 E-7	≈ 6
IBM hex. REAL*8	5.4 E-79	7.2 E 75	2.2 E-16	≈ 15
CRAY X-MP... REAL*8	≈ 1 E-2466	≈ 1 E 2466	≈ 7 E-15	≈ 14

IEEE 754 Rounding:

Compute X•Y as if to infinite precision, and then round to the precision of the destination format as if Range (K) were unlimited (actually requires only three extra bits of precision !).

If this rounded result is too big, OVERFLOW ; default is ±∞ .

If this rounded result is nonzero but too near 0 , UNDERFLOW ; default is to round to nearest finite number even if it is Subnormal.

Mathematical simplicity ...

Finite real numbers all of the simple form $2^{k+1-N} n$

integer n (signed *Significand*)

integer k (unbiased signed *Exponent*)

K+1 Exponent bits: $1 - 2^K < k < 2^K$.

N Significant bits: $-2^N < n < 2^N$.

... vs. Traditional intricacies ...

Normalized nonzero

$2^{k+1-N} n = \pm 2^k (1 + f)$ with a nonnegative *fraction* $f < 1$.

Zero

$\pm 0 = \pm 2^{2-2^K} (0)$ with a sign determinable only by either

CopySign(...) or Division by Zero;

$3/(\pm 0) = \pm \infty$ respectively.

Subnormal (suppressed in prior formats)

$2^{2-2^K} n = \pm 2^{2-2^K}(0 + f)$ with a positive *fraction* $f < 1$ and format's minimum exponent $k = 2 - 2^K$ and $0 < |n| < 2^{N-1}$.
Subnormal numbers can complicate implementation but are needed for

Gradual Underflow:

```
   Subnormals   [--- Normalized Numbers ----- - - -  -  -  -  -   -  -  ->
                |               |                               |
0-!-!-+-!-+-+-+-!-+-+-+-+-+-+-+-!---+---+---+---+---+---+---+---!------ - -
  | |   |       |               |                               |
Powers of 2 :  2^(2-2^K)        2^(3-2^K)                      2^(4-2^K)
```

-+- Consecutive Positive Floating-Point Numbers -+-

Before IEEE 754, a huge empty gap between 0 and the smallest normalized nonzero number exacerbated the problem of distinguishing noxious underflows from the innocuous ones, which are overwhelmingly more numerous.

cf. notes p. 2 and pp. 15-17.

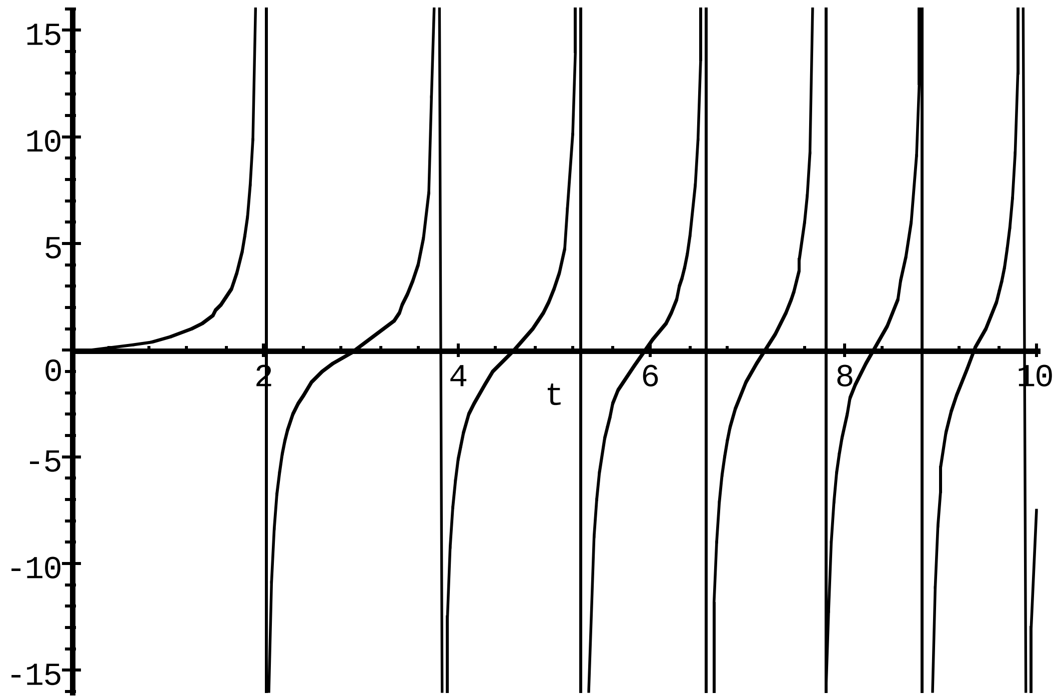
What about ∞ ?

The problem is to compute y(10) where y(t) satisfies the Ricatti equation

$$dy/dt = t + y^2 \quad \text{for all} \quad t \geq 0 , \quad y(0) = 0 .$$

Let us pretend not to know that y(t) may be expressed in terms of Bessel functions J... , whence y(10) = -7.53121 10731 35425 34544 97349 58··· . Instead a numerical method will be used to solve the differential equation approximately and as accurately as desired if enough time is spent on it.

Maple V r3 plots the solution y(t) of a Ricatti equation



. . . .

The function $f(\sigma)$ is a *continued fraction*:

$$f(\sigma) = \sigma - a[n] - \cfrac{b[n]^2}{\sigma - a[n-1] - \cfrac{b[n-1]^2}{\sigma - a[n-2] - \cfrac{b[n-2]^2}{\sigma - a[n-3] - \ldots - \cfrac{b[2]^2}{\sigma - a[1]}}}} .$$