

Floating Point Arithmetic

or

You can't always count on your computer

Derek O'Connor

University College, Dublin

March 4 2005

I do hate sums ... if you add up a sum from the bottom up, and then again from the top down, the result is always different. — *Mrs LaTouche, c. 1875*

- 1 F.P. Number Systems
- 2 Problems with FP Arithmetic
- 3 Intractable Problems?
- 4 Unreliable Software
- 5 War Stories
- 6 Odds and Ends
- 7 Finally

Floating Point representation of the Reals

- Floating Point representation of real numbers
- Machine Epsilon ϵ_m
- Range and Distribution of Floating Point numbers
- IEEE Standard
- Accuracy vs Precision
- Catastrophic Cancellation

Floating Point representation of the Reals

- Floating Point representation of real numbers
- Machine Epsilon ϵ_m
- Range and Distribution of Floating Point numbers
- IEEE Standard
- Accuracy vs Precision
- Catastrophic Cancellation

A floating point number system represents a real number x as

$$\mathbf{fl}(x) = \pm s \times b^e = \pm(0.s_1s_2\dots s_p)_b \times b^e = \pm \left(\frac{s_1}{b^1} + \frac{s_2}{b^2} + \dots + \frac{s_p}{b^p} \right) \times b^e$$

where

- b — is called the **base**.
- s — is called the **significand**,
- p — is the **precision**, the number of digits in the significand,
- e — is called the **exponent**,

$\mathbf{fl}(x)$ is **normalized** if e is adjusted so that $s_1 \neq 0$.

Thus

$$\mathbf{fl}(x) = \left(\sum_{k=1}^p s_k b^{-k} \right) \times b^e \text{ is the f.p. representation of } x = \sum_{k=-\infty}^n d_k b^k$$

$\mathbf{fl}(x)$ is a **finite rational approximation** of x .

A floating point number system represents a real number x as

$$\mathbf{fl}(x) = \pm s \times b^e = \pm(0.s_1s_2\dots s_p)_b \times b^e = \pm \left(\frac{s_1}{b^1} + \frac{s_2}{b^2} + \dots + \frac{s_p}{b^p} \right) \times b^e$$

where

- b — is called the **base**.
- s — is called the **significand**,
- p — is the **precision**, the number of digits in the significand,
- e — is called the **exponent**,

$\mathbf{fl}(x)$ is **normalized** if e is adjusted so that $s_1 \neq 0$.

Thus

$$\mathbf{fl}(x) = \left(\sum_{k=1}^p s_k b^{-k} \right) \times b^e \text{ is the f.p. representation of } x = \sum_{k=-\infty}^n d_k b^k$$

$\mathbf{fl}(x)$ is a **finite rational approximation** of x .

A floating point number system is denoted by $\mathbb{F}(b, p, e_{\min}, e_{\max})$, which is characterized by 4 parameters :

- ① the *base* b ,
- ② the *precision* p ,
- ③ the *minimum exponent* e_{\min} ,
- ④ the *maximum exponent* e_{\max} .

Storage Representation : A floating point number $\pm.s \times b^e$ is represented in a w -digit word as follows :



Note : b is implicit. s and e are represented as integers.

b^w different digit patterns in a w -digit word.

A floating point number system is denoted by $\mathbb{F}(b, p, e_{\min}, e_{\max})$, which is characterized by 4 parameters :

- 1 the *base* b ,
- 2 the *precision* p ,
- 3 the *minimum exponent* e_{\min} ,
- 4 the *maximum exponent* e_{\max} .

Storage Representation : A floating point number $\pm.s \times b^e$ is represented in a w -digit word as follows :

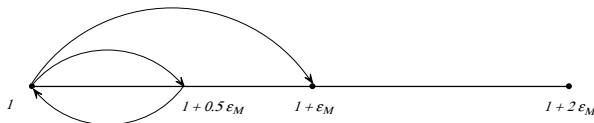


Note : b is implicit. s and e are represented as integers.

b^w different digit patterns in a w -digit word.

The most important derived parameter of a floating point number system is :

Machine Epsilon, ϵ_m : The distance from 1.0 to the next higher floating point number.



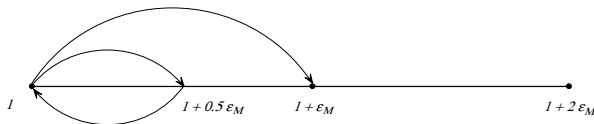
For any $\mathbb{F}(b, p, e_{\min}, e_{\max})$, we have

$$\begin{aligned} \mathbf{fl}(1.0) &= \underbrace{.(100 \dots 00)}_{p \text{ digits}} \times b^1 \\ \mathbf{fl}(1.0 + \epsilon_m) &= \underbrace{.(100 \dots 01)}_{p \text{ digits}} \times b^1 \\ \mathbf{fl}(\epsilon_m) &= \underbrace{.(000 \dots 01)}_{p \text{ digits}} \times b^1 \\ &= b^{1-p} \end{aligned}$$

Machine Epsilon $\epsilon_m = b^{1-p}$

The most important derived parameter of a floating point number system is :

Machine Epsilon, ϵ_m : The distance from 1.0 to the next higher floating point number.



For any $\mathbb{F}(b, p, e_{\min}, e_{\max})$, we have

$$\begin{aligned} \mathbf{fl}(1.0) &= \underbrace{.(100 \dots 00)}_{p \text{ digits}} \times b^1 \\ \mathbf{fl}(1.0 + \epsilon_m) &= \underbrace{.(100 \dots 01)}_{p \text{ digits}} \times b^1 \\ \mathbf{fl}(\epsilon_m) &= \underbrace{.(000 \dots 01)}_{p \text{ digits}} \times b^1 \\ &= b^{1-p} \end{aligned}$$

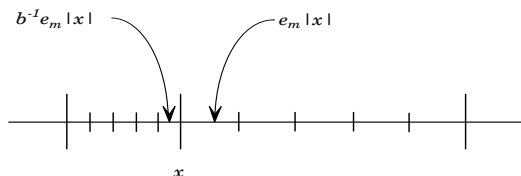
Machine Epsilon $\epsilon_m = b^{1-p}$

Theorem (FLOATING POINT SPACING)

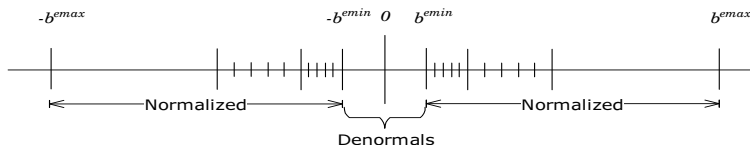
The spacing between a normalized floating point number x and an adjacent normalized number is at least $b^{-1}\epsilon_m|x|$ and at most $\epsilon_m|x|$ (unless x or its neighbour is 0).

This lemma clearly shows that the spacing between adjacent floating point numbers varies with the magnitude of x .

Thus the minimum spacing is $b^{e_{\min}} - p$ and the maximum spacing is $b^{e_{\max}} - p$.



We can see that, in general, when the exponent is small the spacing between floating point numbers is small, and when it is large the spacing is large.



The *density* of floating point numbers is highest around 0 and lowest at either end of the range, i.e., $\pm b^{e_{max}}$. For any given exponent e we have a fixed number of FP numbers between b^e and b^{e+1} . This is b^p and so

$$\text{FP number density is } b^p / b^e = b^{p-e} = cb^{-e}$$

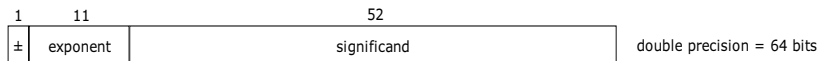
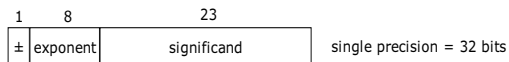
The IEEE Standard defines two floating point arithmetics :

$$\mathbb{F}_{sp}(2, 24, -125, 128), \text{ and } \mathbb{F}_{dp}(2, 53, -1021, 1024).$$

along with

- representation formats.
- operation results.
- rounding modes.
- exception results.

Representation Formats



	Single : F(2, 24, -125, 128)	Double : F(2, 53, -1021, 1024)
Format	[1,8,23]	[1,11,52]
Range	$10^{-38} \leq x \leq 10^{38}$	$10^{-308} \leq x \leq 10^{308}$
Mach Eps	1.2×10^{-7}	2.2×10^{-16}
Precision	6 digits (24 bits)	15 digits (53 bits)

Operation Results : The standard specifies that all arithmetic operations be performed as if they were

- calculated exactly,
- the result normalized,
- rounded according to one of four rounding modes.

$$z = \mathbf{fl}(x \diamond y) = \text{ROUND}(\text{NORMALIZE}(x \diamond y)).$$

Standard FP Model

$$\mathbf{fl}(x \diamond y) = (x \diamond y)(1 + \delta), \text{ where } \diamond \text{ is one of } \{+, -, *, /\}, \text{ and } |\delta| \leq \epsilon_m/2.$$

Rounding Modes : 4 different modes but 'round-to-nearest' is mainly used.

Exception Results :

$\frac{1}{0}$	$\frac{1}{\infty}$	$\frac{0}{0}$	$\frac{\infty}{\infty}$	$\infty \times 0$	1^∞	$\infty - \infty$	0^0	∞^0
inf	0	NaN	NaN	NaN	NaN	NaN	1	1

The IEEE FP 'numbers' `inf` and `NaN` have special properties and are useful for tracking errors in programs.

Example : Parallel Resistors

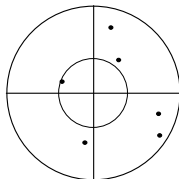
$$R = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

If R_1 is a short circuit, i.e., $R_1 = 0$, then

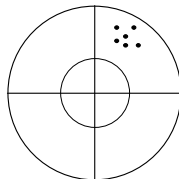
$$R_{sc} = \frac{1}{\frac{1}{0} + \frac{1}{R_2}} = \frac{1}{\infty + \frac{1}{R_2}} = 0$$

It is very important to distinguish between Accuracy and Precision.

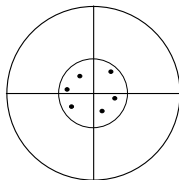
Low Accuracy – Low Precision



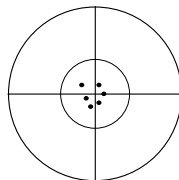
Low Accuracy – High Precision



High Accuracy – Low Precision



High Accuracy – High Precision



Floating point arithmetic is not the same as ‘mathematical’ arithmetic.

Remember:

- $\mathbb{F}(b, p, e_{\min}, e_{\max})$ is a finite subset of \mathbb{Q}
- The f.p. representation of \mathbb{R} has large holes in it.
- \mathbb{F} has a limited range in \mathbb{R} .
- The elements of \mathbb{F} are not uniformly distributed over \mathbb{R} .

Ignorance of these facts give rise to

- Anomalies (apparent).
- ‘Howlers’.
- Junk software.

We first look at a source of many failures in F.P. software —**Catastrophic Cancellation**

Floating point arithmetic is not the same as ‘mathematical’ arithmetic.

Remember:

- $\mathbb{F}(b, p, e_{\min}, e_{\max})$ is a finite subset of \mathbb{Q}
- The f.p. representation of \mathbb{R} has large holes in it.
- \mathbb{F} has a limited range in \mathbb{R} .
- The elements of \mathbb{F} are not uniformly distributed over \mathbb{R} .

Ignorance of these facts give rise to

- Anomalies (apparent).
- ‘Howlers’.
- Junk software.

We first look at a source of many failures in F.P. software —**Catastrophic Cancellation**

Theorem (CANCELLATION ERROR)

In a FP system $\mathbb{F}(b, p, -, -)$ the relative error in $\mathbf{fl}(x - y)$ can be as large as $b - 1$.

This is 100% for $b = 2$ and 900% for $b = 10$.

Proof.

Let $x = .(100 \dots 0) \times b^1$ and $y = .((b - 1)(b - 1) \dots (b - 1)) \times b^0$. The exact difference $(x - y)$ is b^{-t} . When performing the floating point subtraction the digits of y are aligned with x by shifting them to the right. Thus we get

$$\mathbf{fl}(x - y) = .(100 \dots 0) \times b^1 - .(0 (b - 1) \dots (b - 1)) \times b^1 = .(00 \dots 1) \times b^1 = b^{1-p}$$

The relative error is

$$\frac{|(x - y) - \mathbf{fl}(x - y)|}{|x - y|} = \frac{|b^{-p} - b^{1-p}|}{b^{-p}} = |1 - b| = b - 1.$$



Theorem (CANCELLATION ERROR)

In a FP system $\mathbb{F}(b, p, -, -)$ the relative error in $\mathbf{fl}(x - y)$ can be as large as $b - 1$.

This is 100% for $b = 2$ and 900% for $b = 10$.

Proof.

Let $x = .(100 \cdots 0) \times b^1$ and $y = .((b - 1)(b - 1) \cdots (b - 1)) \times b^0$. The exact difference $(x - y)$ is b^{-t} . When performing the floating point subtraction the digits of y are aligned with x by shifting them to the right. Thus we get

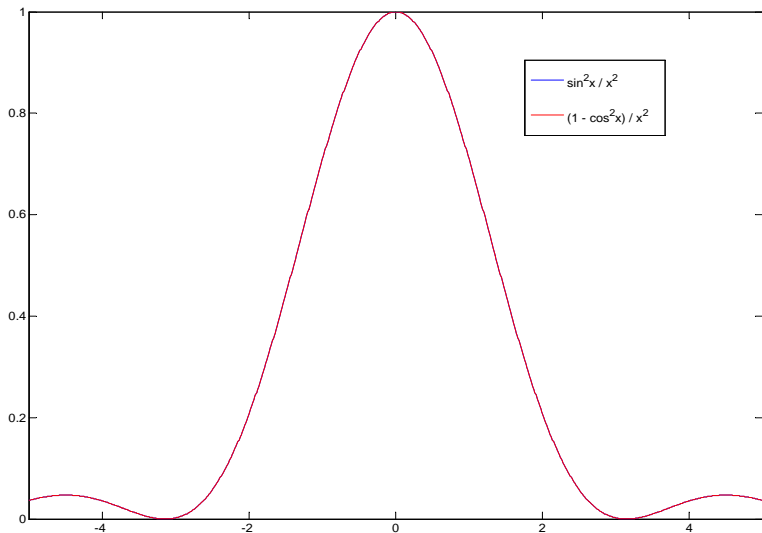
$$\mathbf{fl}(x - y) = .(100 \cdots 0) \times b^1 - .(0 (b - 1) \cdots (b - 1)) \times b^1 = .(00 \cdots 1) \times b^1 = b^{1-p}$$

The relative error is

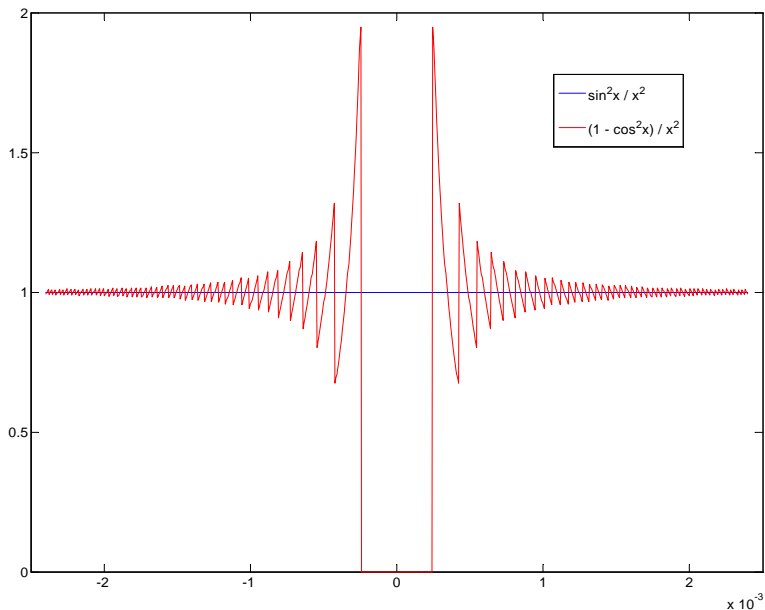
$$\frac{|(x - y) - \mathbf{fl}(x - y)|}{|x - y|} = \frac{|b^{-p} - b^{1-p}|}{b^{-p}} = |1 - b| = b - 1.$$



Example of Catastrophic Cancellation — $\sin^2 x / x^2$ and $(1 - \cos^2 x) / x^2$



Example of Catastrophic Cancellation — $\sin^2 x / x^2$ and $(1 - \cos^2 x) / x^2$



Guard Digits : If we use an **extra digit** when performing subtraction, then instead of

Theorem (CANCELLATION ERROR)

In any floating point system $\mathbb{F}(b, p, -, -)$ the maximum relative error in $\mathbf{fl}(x - y)$ is b^{-1} .

we get

Theorem (CANCELLATION ERROR WITH GUARD DIGIT)

In any floating point system $\mathbb{F}(b, p, -, -)$ the maximum relative error in $\mathbf{fl}(x - y)$ is b^{1-p} .

- IBM in the 1960s had to retro-fit guard digits to its machines.
- CDC and Cray, until recently, had no guard digits. These machines were very fast, but every so often ...

Speed is King

Guard Digits : If we use an **extra digit** when performing subtraction, then instead of

Theorem (CANCELLATION ERROR)

In any floating point system $\mathbb{F}(b, p, -, -)$ the maximum relative error in $\mathbf{fl}(x - y)$ is b^{-1} .

we get

Theorem (CANCELLATION ERROR WITH GUARD DIGIT)

In any floating point system $\mathbb{F}(b, p, -, -)$ the maximum relative error in $\mathbf{fl}(x - y)$ is b^{1-p} .

- IBM in the 1960s had to retro-fit guard digits to its machines.
- CDC and Cray, until recently, had no guard digits. These machines were very fast, but every so often ...

Speed is King

Guard Digits : If we use an **extra digit** when performing subtraction, then instead of

Theorem (CANCELLATION ERROR)

In any floating point system $\mathbb{F}(b, p, -, -)$ the maximum relative error in $\mathbf{fl}(x - y)$ is b^{-1} .

we get

Theorem (CANCELLATION ERROR WITH GUARD DIGIT)

In any floating point system $\mathbb{F}(b, p, -, -)$ the maximum relative error in $\mathbf{fl}(x - y)$ is b^{1-p} .

- IBM in the 1960s had to retro-fit guard digits to its machines.
- CDC and Cray, until recently, had no guard digits. These machines were very fast, but every so often ...

Speed is King

The series $H_n = \sum_{k=1}^n 1/k$ **diverges**. (D'ORESME, 1323–1382)

In finite precision arithmetic $\hat{H}_n = \mathbf{fl}(\sum_{k=1}^n 1/k)$ **converges**.

$$\text{We have } H_n = H_{n-1} + \frac{1}{n},$$

$$\text{and in floating point, } \hat{H}_n = \mathbf{fl}(\hat{H}_{n-1} + \mathbf{fl}(1/n)), \quad \hat{H}_1 = \mathbf{fl}(1) = 1.$$

If $\mathbf{fl}(1/n)$ is zero relative to \hat{H}_{n-1} then $\hat{H}_n = \hat{H}_{n-1}$, and we have convergence.

This happens when $\mathbf{fl}(1/n) < \epsilon_m \hat{H}_{n-1}$, where ϵ_m is machine epsilon.

The series $H_n = \sum_{k=1}^n 1/k$ diverges. (D'ORESME, 1323–1382)

In finite precision arithmetic $\hat{H}_n = \mathbf{fl}(\sum_{k=1}^n 1/k)$ converges.

$$\text{We have } H_n = H_{n-1} + \frac{1}{n},$$

$$\text{and in floating point, } \hat{H}_n = \mathbf{fl}(\hat{H}_{n-1} + \mathbf{fl}(1/n)), \quad \hat{H}_1 = \mathbf{fl}(1) = 1.$$

If $\mathbf{fl}(1/n)$ is zero relative to \hat{H}_{n-1} then $\hat{H}_n = \hat{H}_{n-1}$, and we have convergence.

This happens when $\mathbf{fl}(1/n) < \epsilon_m \hat{H}_{n-1}$, where ϵ_m is machine epsilon.

The series $H_n = \sum_{k=1}^n 1/k$ diverges. (D'ORESME, 1323–1382)

In finite precision arithmetic $\hat{H}_n = \mathbf{fl}(\sum_{k=1}^n 1/k)$ converges.

$$\text{We have } H_n = H_{n-1} + \frac{1}{n},$$

$$\text{and in floating point, } \hat{H}_n = \mathbf{fl}(\hat{H}_{n-1} + \mathbf{fl}(1/n)), \quad \hat{H}_1 = \mathbf{fl}(1) = 1.$$

If $\mathbf{fl}(1/n)$ is zero relative to \hat{H}_{n-1} then $\hat{H}_n = \hat{H}_{n-1}$, and we have convergence.

This happens when $\mathbf{fl}(1/n) < \epsilon_m \hat{H}_{n-1}$, where ϵ_m is machine epsilon.

The series $H_n = \sum_{k=1}^n 1/k$ diverges. (D'ORESME, 1323–1382)

In finite precision arithmetic $\hat{H}_n = \mathbf{fl}(\sum_{k=1}^n 1/k)$ converges.

$$\text{We have } H_n = H_{n-1} + \frac{1}{n},$$

$$\text{and in floating point, } \hat{H}_n = \mathbf{fl}(\hat{H}_{n-1} + \mathbf{fl}(1/n)), \quad \hat{H}_1 = \mathbf{fl}(1) = 1.$$

If $\mathbf{fl}(1/n)$ is zero relative to \hat{H}_{n-1} then $\hat{H}_n = \hat{H}_{n-1}$, and we have convergence.

This happens when $\mathbf{fl}(1/n) < \epsilon_m \hat{H}_{n-1}$, where ϵ_m is machine epsilon.

The series $H_n = \sum_{k=1}^n 1/k$ diverges. (D'ORESME, 1323–1382)

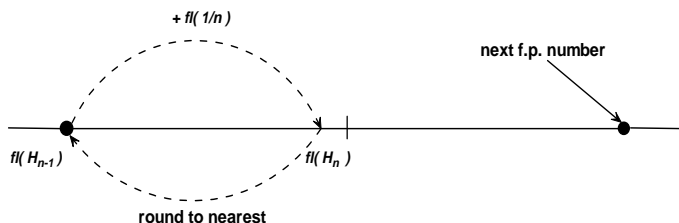
In finite precision arithmetic $\hat{H}_n = \mathbf{fl}(\sum_{k=1}^n 1/k)$ converges.

$$\text{We have } H_n = H_{n-1} + \frac{1}{n},$$

$$\text{and in floating point, } \hat{H}_n = \mathbf{fl}(\hat{H}_{n-1} + \mathbf{fl}(1/n)), \quad \hat{H}_1 = \mathbf{fl}(1) = 1.$$

If $\mathbf{fl}(1/n)$ is zero relative to \hat{H}_{n-1} then $\hat{H}_n = \hat{H}_{n-1}$, and we have convergence.

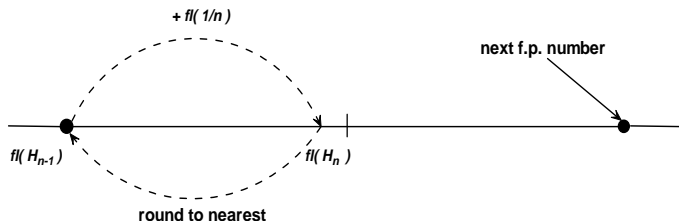
This happens when $\mathbf{fl}(1/n) < \epsilon_m \hat{H}_{n-1}$, where ϵ_m is machine epsilon.



- In IEEE single precision using a Pentium III Xeon, 800MHz machine, convergence to $\hat{H}_n \approx 15$ occurred after $n = 2,097,152$ terms in 1 sec.
- In IEEE double precision the harmonic sum would converge to $\hat{H}_n \approx 35$ after $n \approx 10^{15}$ terms in about $0.5 \times 10^{-6} \times 10^{15} = 5 \times 10^8$ secs ≈ 15 years.
- The first n for which H_n exceeds 100 is

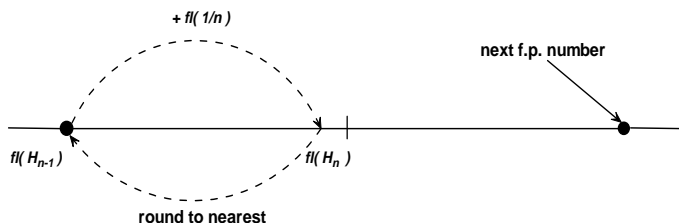
$$n = 15,092,688,622,113,788,323,693,563,264,538,101,449,859,497 \approx 15 \times 10^{42}.$$

H_n approaches ∞ very slowly.



- In IEEE single precision using a Pentium III Xeon, 800MHz machine, convergence to $\hat{H}_n \approx 15$ occurred after $n = 2,097,152$ terms in **1 sec.**
- In IEEE double precision the harmonic sum would converge to $\hat{H}_n \approx 35$ after $n \approx 10^{15}$ terms in about $0.5 \times 10^{-6} \times 10^{15} = 5 \times 10^8$ secs \approx **15 years.**
- The first n for which H_n exceeds 100 is
 $n = 15,092,688,622,113,788,323,693,563,264,538,101,449,859,497 \approx 15 \times 10^{42}$.

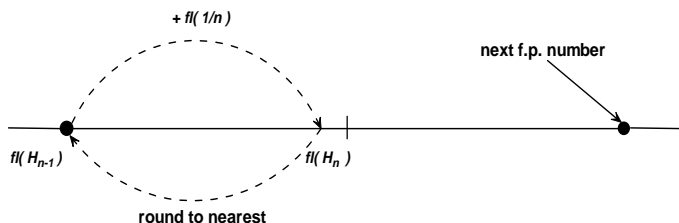
H_n approaches ∞ very slowly.



- In IEEE single precision using a Pentium III Xeon, 800MHz machine, convergence to $\hat{H}_n \approx 15$ occurred after $n = 2,097,152$ terms in **1 sec.**
- In IEEE double precision the harmonic sum would converge to $\hat{H}_n \approx 35$ after $n \approx 10^{15}$ terms in about $0.5 \times 10^{-6} \times 10^{15} = 5 \times 10^8$ secs \approx **15 years.**
- The first n for which H_n exceeds 100 is

$$n = 15,092,688,622,113,788,323,693,563,264,538,101,449,859,497 \approx 15 \times 10^{42}.$$

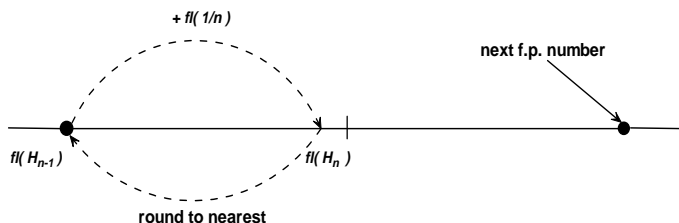
H_n approaches ∞ very slowly.



- In IEEE single precision using a Pentium III Xeon, 800MHz machine, convergence to $\hat{H}_n \approx 15$ occurred after $n = 2,097,152$ terms in **1 sec.**
- In IEEE double precision the harmonic sum would converge to $\hat{H}_n \approx 35$ after $n \approx 10^{15}$ terms in about $0.5 \times 10^{-6} \times 10^{15} = 5 \times 10^8$ secs \approx **15 years.**
- The first n for which H_n exceeds 100 is

$$n = 15,092,688,622,113,788,323,693,563,264,538,101,449,859,497 \approx 15 \times 10^{42}.$$

H_n approaches ∞ very slowly.



- In IEEE single precision using a Pentium III Xeon, 800MHz machine, convergence to $\hat{H}_n \approx 15$ occurred after $n = 2,097,152$ terms in **1 sec.**
- In IEEE double precision the harmonic sum would converge to $\hat{H}_n \approx 35$ after $n \approx 10^{15}$ terms in about $0.5 \times 10^{-6} \times 10^{15} = 5 \times 10^8$ secs \approx **15 years.**
- The first n for which H_n exceeds 100 is

$$n = 15,092,688,622,113,788,323,693,563,264,538,101,449,859,497 \approx 15 \times 10^{42}.$$

H_n approaches ∞ very slowly.

The following 'howlers' are still found in some textbooks.
They are universal in student programs.

❶ if $|x_k - x_{k-1}| < 10^{-6}$ then stop

- If $\text{fl}(x_k) = 10^{50}$ then there is no other floating point number within $\epsilon_m \times 10^{50} = 10^{36}$ of it.
- If this is the only stopping test in a program it will never stop.
- If $\text{fl}(x_k) = 10^{-50}$ then it stops prematurely.
- Solution: if $|x_k - x_{k-1}| < \text{tol} + \epsilon_m |x_k|$ then stop

❷ if $f_k f_{k-1} < 0$ then ... This is a 'clever' version of if $\text{sign}(f_k) \neq \text{sign}(f_{k-1})$

- What happens if $f_k = 10^{-200}$ and $f_{k-1} = -10^{-200}$?
 $f_k f_{k-1} = -10^{-400}$ but $\text{fl}(-10^{-400}) = 0$ (underflow) and the test goes the wrong way.
- If $f_k = 10^{200}$ and $f_{k-1} = -10^{200}$ then we get $\text{fl}(-10^{400})$ (overflow) and the program crashes.

The following 'howlers' are still found in some textbooks.
They are universal in student programs.

❶ if $|x_k - x_{k-1}| < 10^{-6}$ then stop

- If $\text{fl}(x_k) = 10^{50}$ then there is no other floating point number within $\epsilon_m \times 10^{50} = 10^{36}$ of it.
- If this is the only stopping test in a program it will never stop.
- If $\text{fl}(x_k) = 10^{-50}$ then it stops prematurely.
- **Solution:** if $|x_k - x_{k-1}| < \text{tol} + \epsilon_m |x_k|$ then stop

❷ if $f_k f_{k-1} < 0$ then ... This is a 'clever' version of if $\text{sign}(f_k) \neq \text{sign}(f_{k-1})$

- What happens if $f_k = 10^{-200}$ and $f_{k-1} = -10^{-200}$?
 $f_k f_{k-1} = -10^{-400}$ but $\text{fl}(-10^{-400}) = 0$ (underflow) and the test goes the wrong way.
- If $f_k = 10^{200}$ and $f_{k-1} = -10^{200}$ then we get $\text{fl}(-10^{400})$ (overflow) and the program crashes.

The following 'howlers' are still found in some textbooks.
They are universal in student programs.

❶ if $|x_k - x_{k-1}| < 10^{-6}$ then stop

- If $\text{fl}(x_k) = 10^{50}$ then there is no other floating point number within $\epsilon_m \times 10^{50} = 10^{36}$ of it.
- If this is the only stopping test in a program it will never stop.
- If $\text{fl}(x_k) = 10^{-50}$ then it stops prematurely.
- **Solution:** if $|x_k - x_{k-1}| < \text{tol} + \epsilon_m |x_k|$ then stop

❷ if $f_k f_{k-1} < 0$ then ... This is a 'clever' version of if $\text{sign}(f_k) \neq \text{sign}(f_{k-1})$

- What happens if $f_k = 10^{-200}$ and $f_{k-1} = -10^{-200}$?
 $f_k f_{k-1} = -10^{-400}$ but $\text{fl}(-10^{-400}) = 0$ (underflow) and the test goes the wrong way.
- If $f_k = 10^{200}$ and $f_{k-1} = -10^{200}$ then we get $\text{fl}(-10^{400})$ (overflow) and the program crashes.

The following 'howlers' are still found in some textbooks.
They are universal in student programs.

❶ if $|x_k - x_{k-1}| < 10^{-6}$ then stop

- If $\text{fl}(x_k) = 10^{50}$ then there is no other floating point number within $\epsilon_m \times 10^{50} = 10^{36}$ of it.
- If this is the only stopping test in a program it will never stop.
- If $\text{fl}(x_k) = 10^{-50}$ then it stops prematurely.
- **Solution:** if $|x_k - x_{k-1}| < \text{tol} + \epsilon_m |x_k|$ then stop

❷ if $f_k f_{k-1} < 0$ then ... This is a 'clever' version of if $\text{sign}(f_k) \neq \text{sign}(f_{k-1})$

- What happens if $f_k = 10^{-200}$ and $f_{k-1} = -10^{-200}$?
 $f_k f_{k-1} = -10^{-400}$ but $\text{fl}(-10^{-400}) = 0$ (underflow) and the test goes the wrong way.
- If $f_k = 10^{200}$ and $f_{k-1} = -10^{200}$ then we get $\text{fl}(-10^{400})$ (overflow) and the program crashes.

The following 'howlers' are still found in some textbooks.
They are universal in student programs.

❶ if $|x_k - x_{k-1}| < 10^{-6}$ then stop

- If $\mathbf{fl}(x_k) = 10^{50}$ then there is no other floating point number within $\epsilon_m \times 10^{50} = 10^{36}$ of it.
- If this is the only stopping test in a program it will never stop.
- If $\mathbf{fl}(x_k) = 10^{-50}$ then it stops prematurely.
- **Solution:** if $|x_k - x_{k-1}| < \mathbf{tol} + \epsilon_m |x_k|$ then stop

❷ if $f_k f_{k-1} < 0$ then ... This is a 'clever' version of if $\mathbf{sign}(f_k) \neq \mathbf{sign}(f_{k-1})$

- What happens if $f_k = 10^{-200}$ and $f_{k-1} = -10^{-200}$?
 $f_k f_{k-1} = -10^{-400}$ but $\mathbf{fl}(-10^{-400}) = 0$ (underflow) and the test goes the wrong way.
- If $f_k = 10^{200}$ and $f_{k-1} = -10^{200}$ then we get $\mathbf{fl}(-10^{400})$ (overflow) and the program crashes.

The following 'howlers' are still found in some textbooks.
They are universal in student programs.

❶ if $|x_k - x_{k-1}| < 10^{-6}$ then stop

- If $\mathbf{fl}(x_k) = 10^{50}$ then there is no other floating point number within $\epsilon_m \times 10^{50} = 10^{36}$ of it.
- If this is the only stopping test in a program it will never stop.
- If $\mathbf{fl}(x_k) = 10^{-50}$ then it stops prematurely.
- **Solution:** if $|x_k - x_{k-1}| < \mathbf{tol} + \epsilon_m |x_k|$ then stop

❷ if $f_k f_{k-1} < 0$ then ... This is a 'clever' version of if $\mathbf{sign}(f_k) \neq \mathbf{sign}(f_{k-1})$

- What happens if $f_k = 10^{-200}$ and $f_{k-1} = -10^{-200}$?
 $f_k f_{k-1} = -10^{-400}$ but $\mathbf{fl}(-10^{-400}) = 0$ (underflow) and the test goes the wrong way.
- If $f_k = 10^{200}$ and $f_{k-1} = -10^{200}$ then we get $\mathbf{fl}(-10^{400})$ (overflow) and the program crashes.

The following 'howlers' are still found in some textbooks.
They are universal in student programs.

❶ if $|x_k - x_{k-1}| < 10^{-6}$ then stop

- If $\text{fl}(x_k) = 10^{50}$ then there is no other floating point number within $\epsilon_m \times 10^{50} = 10^{36}$ of it.
- If this is the only stopping test in a program it will never stop.
- If $\text{fl}(x_k) = 10^{-50}$ then it stops prematurely.
- **Solution:** if $|x_k - x_{k-1}| < \text{tol} + \epsilon_m |x_k|$ then stop

❷ if $f_k f_{k-1} < 0$ then ... This is a 'clever' version of if $\text{sign}(f_k) \neq \text{sign}(f_{k-1})$

- What happens if $f_k = 10^{-200}$ and $f_{k-1} = -10^{-200}$?
 $f_k f_{k-1} = -10^{-400}$ but $\text{fl}(-10^{-400}) = 0$ (underflow) and the test goes the wrong way.
- If $f_k = 10^{200}$ and $f_{k-1} = -10^{200}$ then we get $\text{fl}(-10^{400})$ (overflow) and the program crashes.

The following 'howlers' are still found in some textbooks.
They are universal in student programs.

❶ if $|x_k - x_{k-1}| < 10^{-6}$ then stop

- If $\mathbf{fl}(x_k) = 10^{50}$ then there is no other floating point number within $\epsilon_m \times 10^{50} = 10^{36}$ of it.
- If this is the only stopping test in a program it will never stop.
- If $\mathbf{fl}(x_k) = 10^{-50}$ then it stops prematurely.
- **Solution:** if $|x_k - x_{k-1}| < \mathbf{tol} + \epsilon_m |x_k|$ then stop

❷ if $f_k f_{k-1} < 0$ then ... This is a 'clever' version of if $\mathbf{sign}(f_k) \neq \mathbf{sign}(f_{k-1})$

- What happens if $f_k = 10^{-200}$ and $f_{k-1} = -10^{-200}$?
 $f_k f_{k-1} = -10^{-400}$ but $\mathbf{fl}(-10^{-400}) = 0$ (underflow) and the test goes the wrong way.
- If $f_k = 10^{200}$ and $f_{k-1} = -10^{200}$ then we get $\mathbf{fl}(-10^{400})$ (overflow) and the program crashes.

The following 'howlers' are still found in some textbooks.
They are universal in student programs.

❶ if $|x_k - x_{k-1}| < 10^{-6}$ then stop

- If $\mathbf{fl}(x_k) = 10^{50}$ then there is no other floating point number within $\epsilon_m \times 10^{50} = 10^{36}$ of it.
- If this is the only stopping test in a program it will never stop.
- If $\mathbf{fl}(x_k) = 10^{-50}$ then it stops prematurely.
- **Solution:** if $|x_k - x_{k-1}| < \text{tol} + \epsilon_m |x_k|$ then stop

❷ if $f_k f_{k-1} < 0$ then ... This is a 'clever' version of if $\text{sign}(f_k) \neq \text{sign}(f_{k-1})$

- What happens if $f_k = 10^{-200}$ and $f_{k-1} = -10^{-200}$?
 $f_k f_{k-1} = -10^{-400}$ but $\mathbf{fl}(-10^{-400}) = 0$ (underflow) and the test goes the wrong way.
- If $f_k = 10^{200}$ and $f_{k-1} = -10^{200}$ then we get $\mathbf{fl}(-10^{400})$ (overflow) and the program crashes.

- Some problems, as they stand, seem to be intractable in standard IEEE arithmetic.
- Here are two, obviously simple problems that do not seem to be solvable in standard IEEE arithmetic.

The NERSC Sea Surface Height Problem.

The following information was taken from a paper by Yun He & Chris Ding of the NERSC-Lawrence Berkeley Labs who were doing a large-scale simulation of ocean circulation.¹

At each step of the simulation the

- *Sea Surface Heights* are calculated at each point on a 64×120 latitude-longitude grid.
- The average of these $64 \times 120 = 7680$ 15-digit numbers is then calculated.
- These averages are calculated for many 64×120 grids and then checked against satellite data.

¹Journal of Supercomputing, Vol 18, No 3, Mar 2001

Here is some standard Fortran code for this problem, along with the general algorithm for summing the elements of the set $X = \{x_1, x_2, \dots, x_n\}$:

Fortran	General
do j = 1, 64	while X > 1 do
do i = 1, 120	(xi,xj) := Delete2(X)
sum = sum + ssh(i,j)	z := xi + xj
end do	Insert(z, X)
end do	end while

In the Fortran code the order of summation is fixed by the do-loops.

In the general algorithm the order is determined by the Delete2(X) function only.

Here are some results that He & Ding got with the Fortran code above, on a single processor, using IEEE double precision (15 digits).

Sum Order	Value (dp)
Longitude First	34.4147682189941410
Reverse Longitude First	32.302734375
Latitude First	0.67326545715332031
Reverse Latitude First	0.734375
Correct Value	0.35798583924770355

Here are the results I got using TMT Pascal's IEEE extended precision (19 digits)

Sum Order	Value (ep)
Latitude First	0.3641029922291636467
Reverse Latitude First	0.3633680343627929687
Correct Value	0.35798583924770355



Here are some results that He & Ding got with the Fortran code above, on a single processor, using IEEE double precision (15 digits).

Sum Order	Value (dp)
Longitude First	34.4147682189941410
Reverse Longitude First	32.302734375
Latitude First	0.67326545715332031
Reverse Latitude First	0.734375
Correct Value	0.35798583924770355

Here are the results I got using TMT Pascal's IEEE extended precision (19 digits)

Sum Order	Value (ep)
Latitude First	0.3641029922291636467
Reverse Latitude First	0.3633680343627929687
Correct Value	0.35798583924770355

Here are some results that He & Ding got with the calculations distributed over 1 to 64 processors, using (1) double precision, (2) self-compensating dp, and (3) Bailey's double-double precision software package.

Sea Surface Height Global Sum

p	Double Precision	SCS	Double-Double	
	MPI_SUM	MPI_SUMDD	MPI_SUM	MPI_SUMDD
1	0.67326545715332031	0.37443733215332031	0.35798583924770355	0.35798583924770355
2	0.59375	0.35705852508544922	0.35798583924770355	0.35798583924770355
4	0.609375	0.41215181350708008	0.35798583924770355	0.35798583924770355
8	0.515625	0.37305688858032227	0.35945068299770355	0.35798583924770355
16	0.3125	0.37309432029724121	0.35993896424770355	0.35798583924770355
32	0.412109375	0.39595553278923035	0.36260925233364105	0.35798583924770355
64	0.404296875	0.40655867755413055	0.35461364686489105	0.35798583924770355

Y. He and C. Ding

Rump's Polynomial.

$$R(x, y) = \frac{33375}{100}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{55}{10}y^8 + \frac{x}{2y}$$

Evaluate $R(x, y)$ at $x = 77617$ and $y = 33096$.

Three precisions were used, with these results :

$$\begin{aligned} \text{sp (6 digs) } R_6 &= 6.33825 \times 10^{29} \\ \text{dp (15 digs) } R_{15} &= 1.1726039400532 \\ \text{ep (35 digs) } R_{35} &= 1.1726039400531786318588349045201838 \end{aligned}$$

Conclusion: $R_{15} = R_{35}$ rounded to 15 digits. R_{15} is correct.

Rump's Polynomial.

$$R(x, y) = \frac{33375}{100}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{55}{10}y^8 + \frac{x}{2y}$$

Evaluate $R(x, y)$ at $x = 77617$ and $y = 33096$.

Three precisions were used, with these results :

$$\begin{aligned} \text{sp (6 digs) } R_6 &= 6.33825 \times 10^{29} \\ \text{dp (15 digs) } R_{15} &= 1.1726039400532 \\ \text{ep (35 digs) } R_{35} &= 1.1726039400531786318588349045201838 \end{aligned}$$

Conclusion: $R_{15} = R_{35}$ rounded to 15 digits. R_{15} is correct.

Rump's Polynomial.

$$R(x, y) = \frac{33375}{100}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{55}{10}y^8 + \frac{x}{2y}$$

Evaluate $R(x, y)$ at $x = 77617$ and $y = 33096$.

Three precisions were used, with these results :

$$\begin{aligned} \text{sp (6 digs) } R_6 &= 6.33825 \times 10^{29} \\ \text{dp (15 digs) } R_{15} &= 1.1726039400532 \\ \text{ep (35 digs) } R_{35} &= 1.1726039400531786318588349045201838 \end{aligned}$$

Conclusion: $R_{15} = R_{35}$ rounded to 15 digits. R_{15} is correct.

Rump's Polynomial.

$$R(x, y) = \frac{33375}{100}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{55}{10}y^8 + \frac{x}{2y}$$

Evaluate $R(x, y)$ at $x = 77617$ and $y = 33096$.

Three precisions were used, with these results :

$$\begin{aligned} \text{sp (6 digs) } R_6 &= 6.33825 \times 10^{29} \\ \text{dp (15 digs) } R_{15} &= 1.1726039400532 \\ \text{ep (35 digs) } R_{35} &= 1.1726039400531786318588349045201838 \end{aligned}$$

Conclusion: $R_{15} = R_{35}$ rounded to 15 digits. R_{15} is correct.

Rump's Polynomial.

$$R(x, y) = \frac{33375}{100}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{55}{10}y^8 + \frac{x}{2y}$$

Evaluate $R(x, y)$ at $x = 77617$ and $y = 33096$.

Three precisions were used, with these results :

$$\begin{aligned} \text{sp (6 digs) } R_6 &= 6.33825 \times 10^{29} \\ \text{dp (15 digs) } R_{15} &= 1.1726039400532 \\ \text{ep (35 digs) } R_{35} &= 1.1726039400531786318588349045201838 \end{aligned}$$

Conclusion: $R_{15} = R_{35}$ rounded to 15 digits. R_{15} is correct.

The exact answer, using rational arithmetic in MAXIMA , is

$$R(77617, 33096) = -\frac{54767}{66192}$$

This rational has the following decimal expansion, with pre-period of length 4 and period of length 294 :

-0.8273

```

96059 94682 13681 41165 09547 98162 91999 03311 57843 84819 91781 48416 72709
69301 42615 42180 32390 62122 31085 32753 20280 39642 25284 02223 83369 59149
14189 02586 41527 67706 06719 84529 85255 01571 18685 03746 67633 55088 22818
46748 85182 49939 56973 65240 51244 86342 76045 44355 81339 13463 86270 24413
82644 42832 97075 17524 77640 80251 3898
96059 ...

```

Conclusion: R_{15} , R_{35} totally wrong. Increasing precision can be misleading.

Solution:

- None if 15 digits used — 40 digits precision needed to get correct answer.
- Rational Arithmetic — expensive 10 – 100 times slower.
- Interval Arithmetic — ditto.
- Re-write expression — yes, but how?

The exact answer, using rational arithmetic in MAXIMA , is

$$R(77617, 33096) = -\frac{54767}{66192}$$

This rational has the following decimal expansion, with pre-period of length 4 and period of length 294 :

-0.8273

```

96059 94682 13681 41165 09547 98162 91999 03311 57843 84819 91781 48416 72709
69301 42615 42180 32390 62122 31085 32753 20280 39642 25284 02223 83369 59149
14189 02586 41527 67706 06719 84529 85255 01571 18685 03746 67633 55088 22818
46748 85182 49939 56973 65240 51244 86342 76045 44355 81339 13463 86270 24413
82644 42832 97075 17524 77640 80251 3898
96059 ...

```

Conclusion: R_{15} , R_{35} totally wrong. Increasing precision can be misleading.

Solution:

- None if 15 digits used — 40 digits precision needed to get correct answer.
- Rational Arithmetic — expensive 10 – 100 times slower.
- Interval Arithmetic — ditto.
- Re-write expression — yes, but how?

The exact answer, using rational arithmetic in MAXIMA , is

$$R(77617, 33096) = -\frac{54767}{66192}$$

This rational has the following decimal expansion, with pre-period of length 4 and period of length 294 :

-0.8273

```

96059 94682 13681 41165 09547 98162 91999 03311 57843 84819 91781 48416 72709
69301 42615 42180 32390 62122 31085 32753 20280 39642 25284 02223 83369 59149
14189 02586 41527 67706 06719 84529 85255 01571 18685 03746 67633 55088 22818
46748 85182 49939 56973 65240 51244 86342 76045 44355 81339 13463 86270 24413
82644 42832 97075 17524 77640 80251 3898
96059 ...

```

Conclusion: R_{15} , R_{35} totally wrong. Increasing precision can be misleading.

Solution:

- None if 15 digits used — 40 digits precision needed to get correct answer.
- Rational Arithmetic — expensive 10 – 100 times slower.
- Interval Arithmetic — ditto.
- Re-write expression — yes, but how?

- 1 Compilers
- 2 Mathematical Systems : Symbolic & Numeric
- 3 Specific
- 4 General

- 1 Compilers
- 2 Mathematical Systems : Symbolic & Numeric
- 3 Specific
- 4 General

- 1 Compilers
- 2 Mathematical Systems : Symbolic & Numeric
- 3 Specific
- 4 General

- 1 Compilers
- 2 Mathematical Systems : Symbolic & Numeric
- 3 Specific
- 4 General

AMD and Intel processors conform to IEEE standard

- Format is correct, along with guard digits, sticky bits, etc.
- The operations $+$, $-$, \times , $/$, $1/x$, $\sqrt{\quad}$, \sin , \cos are carefully implemented in hardware.

Many compilers do not conform to IEEE standard

- Format may be incorrect.
- Do not allow access to hardware features.
- Do not handle exceptions correctly.
- Do not use hardware \sin , \cos correctly.
- etc.

We had to write our own \sin and \cos because Microsoft's C compiler did not reduce the arguments so that they could be handled by the processor. — CLEVE MOLER, Matlab inventor

AMD and Intel processors conform to IEEE standard

- Format is correct, along with guard digits, sticky bits, etc.
- The operations $+$, $-$, \times , $/$, $1/x$, $\sqrt{\quad}$, \sin , \cos are carefully implemented in hardware.

Many compilers do not conform to IEEE standard

- Format may be incorrect.
- Do not allow access to hardware features.
- Do not handle exceptions correctly.
- Do not use hardware \sin , \cos correctly.
- etc.

We had to write our own \sin and \cos because Microsoft's C compiler did not reduce the arguments so that they could be handled by the processor. — CLEVE MOLER, Matlab inventor

AMD and Intel processors conform to IEEE standard

- Format is correct, along with guard digits, sticky bits, etc.
- The operations $+$, $-$, \times , $/$, $1/x$, $\sqrt{\quad}$, \sin , \cos are carefully implemented in hardware.

Many compilers do not conform to IEEE standard

- Format may be incorrect.
- Do not allow access to hardware features.
- Do not handle exceptions correctly.
- Do not use hardware \sin , \cos correctly.
- etc.

We had to write our own \sin and \cos because Microsoft's C compiler did not reduce the arguments so that they could be handled by the processor. — CLEVE MOLER, Matlab inventor

Calculating Elementary Functions is not easy

TABLE V. $\sin(10^{22})$ AND $\cos(10^{22})$ ON VARIOUS SYSTEMS

Computer	$\sin(10^{22})$	$\cos(10^{22})$
Maple 8 (15 digs)	-0.852200849 ...	+0.523214785 ...
Maxima 5.9 (bfloat)	-0.852200849 ...	+0.523214785 ...
Matlab 6.5 (15 digs)	-0.852200849 ...	+0.523214785 ...
O-Matrix 5.5(e format)	+0.226946577 ...	-0.973907232 ...
O-Matrix 5.5(d format)	+0.412143367 ...	-0.911119007 ...
Scilab 3.0 (15 digs)	$+10^{22}$	$+10^{22}$
DVF 5.0 D (sp)	+0.2269466 ...	-0.9739072 ...
DVF 5.0 D (dp)	+0.412143367 ...	-0.911119007 ...
Intel Fortran 8 (sp)	$+9.9999998 \times 10^{21}$	$+9.9999998 \times 10^{21}$
Intel Fortran 8 (dp)	$+10^{22}$	$+10^{22}$
Intel Fortran 8 (ep)	-0.852200849 ...	+0.523214785 ...
Watfor 11.2 (sp)	+0.2812271 ...	-0.9596413 ...
Watfor 11.2 (dp)	+0.4626130 ...	-0.8865603 ...
TMT Pascal(all prec)	+0.0	+0.0
FranzLisp	+0.2269465 ...	-0.9739072 ...
Sharp EL-531VH	error2	error2
MS Windows Calc (32 digs)	-0.852200849 ...	+0.523214785 ...
PariGP 2.2.7 (28 digs)	-0.852200849 ...	+0.523214785 ...
Correct Answer	-0.852200849 ...	+0.523214785 ...

Mathematica 2.0 and the 'New Paradigm for Technical Computing'

by STEPHEN WOLFRAM

or

A Wolf [ram] in the Lions' Den

Recorded by Richard Fateman, Berkeley, 1991.

- Prof. W. Kahan spoke up²
He asked SW to type 12 characters into MATHEMATICA. After some prompting from the audience, SW overcame his reluctance.
$$(1/3)^x + 3^x$$
- Kahan challenged him to simplify it.
MATHEMATICA couldn't, but SW inserted some values for x, and each time got "1" or in the case of a complex number, nearly 1.
- Kahan pointed out that for any real or complex value for x, this expression is 1, but MATHEMATICA doesn't know it.
SW said you could write a pattern and asked Kahan...
"What's your point?"

²Kahan earlier distributed a page of problems MATHEMATICA 1.2 botches.

Mathematica 2.0 and the 'New Paradigm for Technical Computing'

by STEPHEN WOLFRAM

or

A Wolf [ram] in the Lions' Den

Recorded by Richard Fateman, Berkeley, 1991.

- Prof. W. **Kahan** spoke up²
He asked **SW** to type 12 characters into MATHEMATICA. After some prompting from the audience, **SW** overcame his reluctance.

$$(1/3)^x * 3^x$$

- **Kahan** challenged him to simplify it.
MATHEMATICA couldn't, but **SW** inserted some values for x , and each time got "1" or in the case of a complex number, nearly 1.
- **Kahan** pointed out that for any real or complex value for x , this expression is 1, but MATHEMATICA doesn't know it.
SW said you could write a pattern and asked **Kahan**...
"What's your point?"

²Kahan earlier distributed a page of problems MATHEMATICA 1.2 botches.

Mathematica 2.0 and the 'New Paradigm for Technical Computing'

by STEPHEN WOLFRAM

or

A Wolf [ram] in the Lions' Den

Recorded by Richard Fateman, Berkeley, 1991.

- Prof. W. **Kahan** spoke up²
He asked **SW** to type 12 characters into MATHEMATICA. After some prompting from the audience, **SW** overcame his reluctance.
$$(1/3)^x * 3^x$$
- **Kahan** challenged him to simplify it.
MATHEMATICA couldn't, but **SW** inserted some values for x , and each time got "1" or in the case of a complex number, nearly 1.
- **Kahan** pointed out that for any real or complex value for x , this expression is 1, but MATHEMATICA doesn't know it.
SW said you could write a pattern and asked **Kahan**...
"What's your point?"

²Kahan earlier distributed a page of problems MATHEMATICA 1.2 botches.

Mathematica 2.0 and the 'New Paradigm for Technical Computing'

by STEPHEN WOLFRAM

or

A Wolf [ram] in the Lions' Den

Recorded by Richard Fateman, Berkeley, 1991.

- Prof. W. **Kahan** spoke up²
He asked **SW** to type 12 characters into MATHEMATICA. After some prompting from the audience, **SW** overcame his reluctance.
$$(1/3)^x * 3^x$$
- **Kahan** challenged him to simplify it.
MATHEMATICA couldn't, but **SW** inserted some values for x , and each time got "1" or in the case of a complex number, nearly 1.
- **Kahan** pointed out that **for any real or complex value for x , this expression is 1**, but MATHEMATICA doesn't know it.
SW said you could write a pattern and asked **Kahan**...
"What's your point?"

²Kahan earlier distributed a page of problems MATHEMATICA 1.2 botches.

- **Kahan** then asked **SW** to type the same expression but with r instead of 3. With such a substitution, `PowerExpand` in MATHEMATICA reduces the expression for arbitrary r and x , to 1.
- **Kahan** pointed out that for arbitrary r , this expression is **not** 1.
- **SW** repeated *"What's your point?"*
- **Kahan** said that he thought MATHEMATICA was deficient in its capabilities ... That it was **"built on a foundation of sand"**.
- **SW** asked, *"Does anyone else want to make a speech?"*

..... Since my ride home was leaving, I left at this point... I understand the rest of the questions were of the sort "have you fixed this bug" , etc.³

³Richard Fateman, PhD(App. Math.), Harvard 1971, one of the main authors of MACSYMA. Now available free as MAXIMA.

- **Kahan** then asked **SW** to type the same expression but with r instead of 3. With such a substitution, `PowerExpand` in `MATHEMATICA` reduces the expression for arbitrary r and x , to 1.
- **Kahan** pointed out that for arbitrary r , this expression is **not** 1.
- *SW* repeated *"What's your point?"*
- **Kahan** said that he thought `MATHEMATICA` was deficient in its capabilities ... That it was **"built on a foundation of sand"**.
- *SW* asked, *"Does anyone else want to make a speech?"*

..... Since my ride home was leaving, I left at this point... I understand the rest of the questions were of the sort "have you fixed this bug" , etc.³

³Richard Fateman, PhD(App. Math.), Harvard 1971, one of the main authors of `MACSYMA`. Now available free as `MAXIMA`.

- **Kahan** then asked **SW** to type the same expression but with r instead of 3. With such a substitution, `PowerExpand` in MATHEMATICA reduces the expression for arbitrary r and x , to 1.
- **Kahan** pointed out that for arbitrary r , this expression is **not** 1.
- **SW** repeated *"What's your point?"*
- **Kahan** said that he thought MATHEMATICA was deficient in its capabilities ... That it was *"built on a foundation of sand"*.
- **SW** asked, *"Does anyone else want to make a speech?"*

..... Since my ride home was leaving, I left at this point... I understand the rest of the questions were of the sort "have you fixed this bug" , etc.³

³Richard Fateman, PhD(App. Math.), Harvard 1971, one of the main authors of MACSYMA. Now available free as MAXIMA.

- **Kahan** then asked **SW** to type the same expression but with r instead of 3. With such a substitution, `PowerExpand` in `MATHEMATICA` reduces the expression for arbitrary r and x , to 1.
- **Kahan** pointed out that for arbitrary r , this expression is **not** 1.
- **SW** repeated *"What's your point?"*
- **Kahan** said that he thought `MATHEMATICA` was deficient in its capabilities ... That it was **"built on a foundation of sand"**.
- **SW** asked, *"Does anyone else want to make a speech?"*

..... Since my ride home was leaving, I left at this point... I understand the rest of the questions were of the sort "have you fixed this bug" , etc.³

³Richard Fateman, PhD(App. Math.), Harvard 1971, one of the main authors of `MACSYMA`. Now available free as `MAXIMA`.

- **Kahan** then asked **SW** to type the same expression but with r instead of 3. With such a substitution, `PowerExpand` in `MATHEMATICA` reduces the expression for arbitrary r and x , to 1.
- **Kahan** pointed out that for arbitrary r , this expression is **not** 1.
- **SW** repeated *"What's your point?"*
- **Kahan** said that he thought `MATHEMATICA` was deficient in its capabilities ... That it was **"built on a foundation of sand"**.
- **SW** asked, *"Does anyone else want to make a speech?"*

..... Since my ride home was leaving, I left at this point... I understand the rest of the questions were of the sort "have you fixed this bug" , etc.³

³Richard Fateman, PhD(App. Math.), Harvard 1971, one of the main authors of `MACSYMA`. Now available free as `MAXIMA`.

- **Kahan** then asked **SW** to type the same expression but with r instead of 3. With such a substitution, `PowerExpand` in `MATHEMATICA` reduces the expression for arbitrary r and x , to 1.
- **Kahan** pointed out that for arbitrary r , this expression is **not** 1.
- **SW** repeated *"What's your point?"*
- **Kahan** said that he thought `MATHEMATICA` was deficient in its capabilities ... That it was **"built on a foundation of sand"**.
- **SW** asked, *"Does anyone else want to make a speech?"*

..... Since my ride home was leaving, I left at this point... I understand the rest of the questions were of the sort "have you fixed this bug" , etc.³

³Richard Fateman, PhD(App. Math.), Harvard 1971, one of the main authors of `MACSYMA`. Now available free as `MAXIMA`.

- **Kahan** then asked **SW** to type the same expression but with r instead of 3. With such a substitution, `PowerExpand` in `MATHEMATICA` reduces the expression for arbitrary r and x , to 1.
 - **Kahan** pointed out that for arbitrary r , this expression is **not** 1.
 - **SW** repeated *"What's your point?"*
 - **Kahan** said that he thought `MATHEMATICA` was deficient in its capabilities ... That it was **"built on a foundation of sand"**.
 - **SW** asked, *"Does anyone else want to make a speech?"*
-

.... Since my ride home was leaving, I left at this point... I understand the rest of the questions were of the sort "have you fixed this bug" , etc.³

³Richard Fateman, PhD(App. Math.), Harvard 1971, one of the main authors of `MACSYMA`. Now available free as `MAXIMA`.

- ① D.H. Lehmer of Berkeley invented the *multiplicative congruential* algorithm in the late 1940's. Many random number generators in use today are based on Lehmer's ideas.

$$x_{k+1} = (ax_k + c) \pmod{m}.$$

These generators are defined by three integer parameters, a , c , and m . An initial value, x_0 , called the *seed*, starts the generator.

- The parameters a , c , and m . must be chosen with great care so that they generate a sequence of uniformly distributed integers.
- ② In the early 1960's the IBM 360 series came with a random number generator called RANDU which had $a = 65539 = 2^{16} + 3$, $c = 0$, and $m = 2^{31}$. Hence they got

$$x_{k+1} = (2^{16} + 3)x_k \pmod{2^{31}}.$$

- Arithmetic $\pmod{2^{31}}$ can be done quickly.
- $(2^{16} + 3)x_k$ can be done with a shift and an addition.

This was to be a really fast generator

③

- 1 D.H. Lehmer of Berkeley invented the *multiplicative congruential* algorithm in the late 1940's. Many random number generators in use today are based on Lehmer's ideas.

$$x_{k+1} = (ax_k + c) \pmod{m}.$$

These generators are defined by three integer parameters, a , c , and m . An initial value, x_0 , called the *seed*, starts the generator.

- The parameters a , c , and m . must be chosen with great care so that they generate a sequence of uniformly distributed integers.
- 2 In the early 1960's the IBM 360 series came with a random number generator called RANDU which had $a = 65539 = 2^{16} + 3$, $c = 0$, and $m = 2^{31}$. Hence they got

$$x_{k+1} = (2^{16} + 3)x_k \pmod{2^{31}}.$$

- Arithmetic $\pmod{2^{31}}$ can be done quickly.
- $(2^{16} + 3)x_k$ can be done with a shift and an addition.

This was to be a really fast generator

3

- ① D.H. Lehmer of Berkeley invented the *multiplicative congruential* algorithm in the late 1940's. Many random number generators in use today are based on Lehmer's ideas.

$$x_{k+1} = (ax_k + c) \pmod{m}.$$

These generators are defined by three integer parameters, a , c , and m . An initial value, x_0 , called the *seed*, starts the generator.

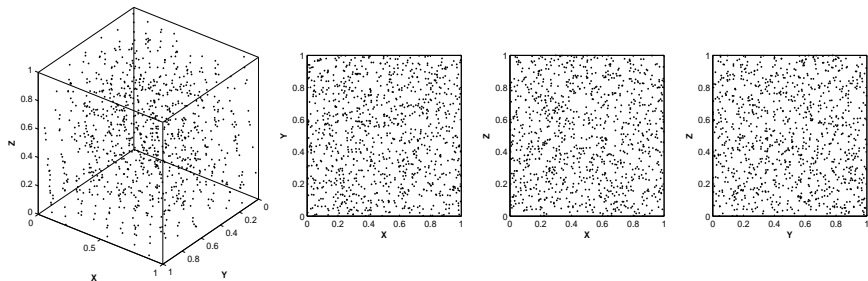
- The parameters a , c , and m . must be chosen with great care so that they generate a sequence of uniformly distributed integers.
- ② In the early 1960's the IBM 360 series came with a random number generator called RANDU which had $a = 65539 = 2^{16} + 3$, $c = 0$, and $m = 2^{31}$. Hence they got

$$x_{k+1} = (2^{16} + 3)x_k \pmod{2^{31}}.$$

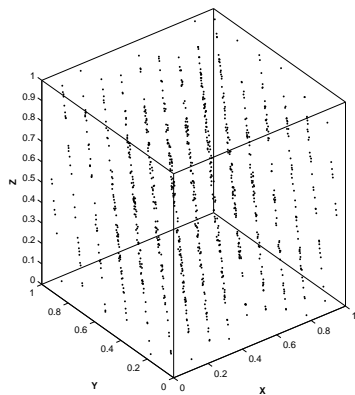
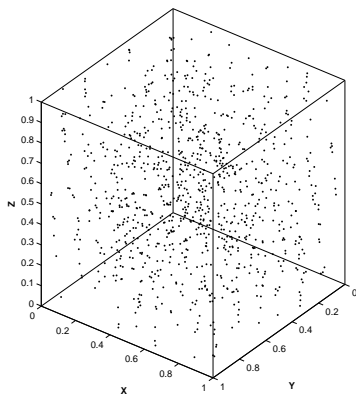
- Arithmetic $\pmod{2^{31}}$ can be done quickly.
- $(2^{16} + 3)x_k$ can be done with a shift and an addition.

This was to be a really fast generator

IBM 360's Randu generator



All views seem random.



A rotation reveals the planes

All P-R Generators have a lattice structure in higher dimensions.

Analysis of Randu : $x_{k+1} = (2^{16} + 3)x_k \pmod{2^{31}}$.

$$\begin{aligned}
 x_{k+2} &= (2^{16} + 3)x_{k+1} \\
 &= (2^{16} + 3)^2 x_k \\
 &= (2^{32} + 6 \cdot 2^{16} + 9)x_k \\
 &= [6 \cdot (2^{16} + 3) - 9]x_k \\
 &= 6 \cdot (2^{16} + 3)x_k - 9x_k \\
 x_{k+2} &= (6x_{k+1} - 9x_k) \pmod{2^{31}}
 \end{aligned}$$

THREE SUCCESSIVE RANDOM NUMBERS ARE HIGHLY CORRELATED.

The last 15 years' simulation studies should be thrown out — Bennett Fox, 1981.

Recommendation : The Mersenne Twister

All P-R Generators have a lattice structure in higher dimensions.

Analysis of Randu : $x_{k+1} = (2^{16} + 3)x_k \pmod{2^{31}}$.

$$\begin{aligned}
 x_{k+2} &= (2^{16} + 3)x_{k+1} \\
 &= (2^{16} + 3)^2 x_k \\
 &= (2^{32} + 6 \cdot 2^{16} + 9)x_k \\
 &= [6 \cdot (2^{16} + 3) - 9]x_k \\
 &= 6 \cdot (2^{16} + 3)x_k - 9x_k \\
 x_{k+2} &= (6x_{k+1} - 9x_k) \pmod{2^{31}}
 \end{aligned}$$

THREE SUCCESSIVE RANDOM NUMBERS ARE HIGHLY CORRELATED.

The last 15 years' simulation studies should be thrown out — Bennett Fox, 1981.

Recommendation : The Mersenne Twister

All P-R Generators have a lattice structure in higher dimensions.

Analysis of Randu : $x_{k+1} = (2^{16} + 3)x_k \pmod{2^{31}}$.

$$\begin{aligned}
 x_{k+2} &= (2^{16} + 3)x_{k+1} \\
 &= (2^{16} + 3)^2 x_k \\
 &= (2^{32} + 6 \cdot 2^{16} + 9)x_k \\
 &= [6 \cdot (2^{16} + 3) - 9]x_k \\
 &= 6 \cdot (2^{16} + 3)x_k - 9x_k \\
 x_{k+2} &= (6x_{k+1} - 9x_k) \pmod{2^{31}}
 \end{aligned}$$

THREE SUCCESSIVE RANDOM NUMBERS ARE HIGHLY CORRELATED.

The last 15 years' simulation studies should be thrown out — Bennett Fox, 1981.

Recommendation : The Mersenne Twister

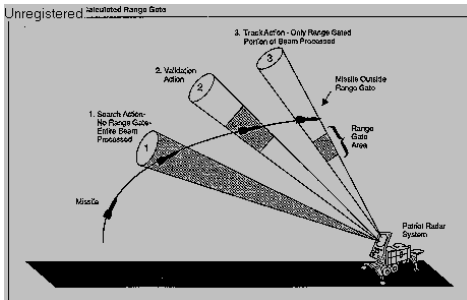
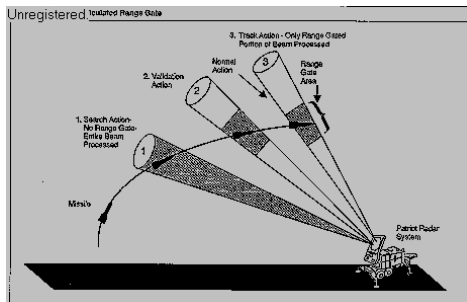
- 1 Patriot Missile.
- 2 Vancouver Stock Exchange.
- 3 Green Party in Schleswig-Holstein.

- 1 Patriot Missile.
- 2 Vancouver Stock Exchange.
- 3 Green Party in Schleswig-Holstein.

- 1 Patriot Missile.
- 2 Vancouver Stock Exchange.
- 3 Green Party in Schleswig-Holstein.

On February 25, 1991, a Patriot missile defense system operating at Dhahran, Saudi Arabia, during Operation Desert Storm failed to track and intercept an incoming Scud. This Scud subsequently hit an Army barracks, killing 28 Americans.





Report of US GAO, Feb 4 1992

Page 14 -- GAO/IMTEC-92-26 Patriot Missile Software Problem -- Appendix II

Effect of Extended Run Time on Patriot Operation

Hours	Seconds	Calculated Time (Seconds)	Inaccuracy (Seconds)	Approximate Shift In Range Gate (Meters)
0	0	0	0	0
1	3600	3599.9966	.0034	7
8	28800	28799.9725	.0025	55
--> 20(a)	72000	71999.9313	.0687	137 <--
48	172800	172799.8352	.1648	330
72	259200	259199.7528	.2472	494
100(b)	360000	359999.6667	.3433	687

- a. Continuous operation exceeding about 20 hours--target outside range gate.
 b. Alpha Battery ran continuously for about 100 hours.

Answer : Range gate shifted because clock was inaccurate.

Note : Relative Clock Error = $|720000 - 71999.9313|/720000 = 0.0687/720000 \approx 10^{-7}$.

Question : Why was the clock inaccurate?

Report of US GAO, Feb 4 1992

Page 14 -- GAO/IMTEC-92-26 Patriot Missile Software Problem -- Appendix II

Effect of Extended Run Time on Patriot Operation

Hours	Seconds	Calculated Time (Seconds)	Inaccuracy (Seconds)	Approximate Shift In Range Gate (Meters)
0	0	0	0	0
1	3600	3599.9966	.0034	7
8	28800	28799.9725	.0025	55
--> 20(a)	72000	71999.9313	.0687	137 <--
48	172800	172799.8352	.1648	330
72	259200	259199.7528	.2472	494
100(b)	360000	359999.6667	.3433	687

a. Continuous operation exceeding about 20 hours--target outside range gate.

b. Alpha Battery ran continuously for about 100 hours.

Answer : Range gate shifted because clock was inaccurate.

Note : Relative Clock Error = $|720000 - 71999.9313|/720000 = 0.0687/720000 \approx 10^{-7}$.

Question : Why was the clock inaccurate?

Report of US GAO, Feb 4 1992

Page 14 -- GAO/IMTEC-92-26 Patriot Missile Software Problem -- Appendix II

Effect of Extended Run Time on Patriot Operation

Hours	Seconds	Calculated Time (Seconds)	Inaccuracy (Seconds)	Approximate Shift In Range Gate (Meters)
0	0	0	0	0
1	3600	3599.9966	.0034	7
8	28800	28799.9725	.0025	55
--> 20(a)	72000	71999.9313	.0687	137 <--
48	172800	172799.8352	.1648	330
72	259200	259199.7528	.2472	494
100(b)	360000	359999.6667	.3433	687

a. Continuous operation exceeding about 20 hours--target outside range gate.

b. Alpha Battery ran continuously for about 100 hours.

Answer : Range gate shifted because clock was inaccurate.

Note : Relative Clock Error = $|720000 - 71999.9313|/720000 = 0.0687/720000 \approx 10^{-7}$.

Question : Why was the clock inaccurate?

Here is an incomplete explanation of what went wrong :

- The internal clock kept time as an integer value in units of 1/10 second.
- Computer clock updated using `CTime := CTime + 0.1`
- Computer's registers were only 24 bits long.
- $1/10 = (0.00010011001100\dots)_2$ and $\text{fl}(1/10) = 0.1 \times (1 - 2^{-20})$.

This leads to a relative error in the computer time of 2^{-20} .

- After 20 hours the error was $20 * 60 * 60 \times 2^{-20} = 0.0686645508$ secs.
- After 100 hours the error was $100 * 60 * 60 \times 2^{-20} = 0.343322754$ secs.
- A Scud missile travels $3750/(3*60*60) = 0.35$ miles in 1/3 sec.

Quick Fix : Reboot the computer every 8 hours — takes 90 secs.

Windows 95 users would have rebooted every 10 mins.

Here is an incomplete explanation of what went wrong :

- The internal clock kept time as an integer value in units of 1/10 second.
- Computer clock updated using `CTime := CTime + 0.1`
- Computer's registers were only 24 bits long.
- $1/10 = (0.00010011001100\dots)_2$ and $\text{fl}(1/10) = 0.1 \times (1 - 2^{-20})$.

This leads to a relative error in the computer time of 2^{-20} .

- After 20 hours the error was $20 * 60 * 60 \times 2^{-20} = 0.0686645508$ secs.
- After 100 hours the error was $100 * 60 * 60 \times 2^{-20} = 0.343322754$ secs.
- A Scud missile travels $3750/(3*60*60) = 0.35$ miles in 1/3 sec.

Quick Fix : Reboot the computer every 8 hours — takes 90 secs.

Windows 95 users would have rebooted every 10 mins.

Here is an incomplete explanation of what went wrong :

- The internal clock kept time as an integer value in units of 1/10 second.
- Computer clock updated using `CTime := CTime + 0.1`
- Computer's registers were only 24 bits long.
- $1/10 = (0.00010011001100\dots)_2$ and $\text{fl}(1/10) = 0.1 \times (1 - 2^{-20})$.

This leads to a relative error in the computer time of 2^{-20} .

- After 20 hours the error was $20 * 60 * 60 \times 2^{-20} = 0.0686645508$ secs.
- After 100 hours the error was $100 * 60 * 60 \times 2^{-20} = 0.343322754$ secs.
- A Scud missile travels $3750 / (3 * 60 * 60) = 0.35$ miles in 1/3 sec.

Quick Fix : Reboot the computer every 8 hours — takes 90 secs.

Windows 95 users would have rebooted every 10 mins.

Here is an incomplete explanation of what went wrong :

- The internal clock kept time as an integer value in units of 1/10 second.
- Computer clock updated using `CTime := CTime + 0.1`
- Computer's registers were only 24 bits long.
- $1/10 = (0.00010011001100\dots)_2$ and $\text{fl}(1/10) = 0.1 \times (1 - 2^{-20})$.

This leads to a relative error in the computer time of 2^{-20} .

- After 20 hours the error was $20 * 60 * 60 \times 2^{-20} = 0.0686645508$ secs.
- After 100 hours the error was $100 * 60 * 60 \times 2^{-20} = 0.343322754$ secs.
- A Scud missile travels $3750/(3*60*60) = 0.35$ miles in 1/3 sec.

Quick Fix : Reboot the computer every 8 hours — takes 90 secs.

Windows 95 users would have rebooted every 10 mins.

- The Patriot system was originally designed (1960's) to operate in Europe against Soviet medium- to high-altitude aircraft and cruise missiles traveling at speeds up to about MACH 2 (1500 mph).
- To avoid detection it was designed to be mobile and *operate for only a few hours* at one location
- Patriot battalions were deployed in Saudi Arabia and then Israel to defend against SCUD missiles (1950's Russian technology).
- Scud missiles fly at approximately MACH 5 (3750 mph)
- Israelis used data-loggers and soon reported tracking errors to US.
-

We finish with two mathematical oddities :

- 1 Newcomb-Borel Paradox.
- 2 Calculation of π .

We finish with two mathematical oddities :

- 1 Newcomb-Borel Paradox.
- 2 Calculation of π .

Emile Borel, in 1909, proved that **almost all real numbers are normal** : in any base, the first digit and all subsequent digits are **uniformly distributed** . For example, with $b = 10$,

$$\Pr(\text{first significant digit} = d) = \frac{1}{9}, \quad d = 1, \dots, 9.$$

However, almost 30 years before, the astronomer **Simon Newcomb**⁴ had proved *Newcomb's Law* : **most natural numbers are logarithmically distributed** :

$$\Pr(\text{first significant digit} = d) = \log_{10}\left(1 + \frac{1}{d}\right), \quad d = 1, \dots, 9,$$

where **Natural Numbers** are those obtained from counting, measurement, or calculation. This gives the following distribution :

d	1	2	3	4	5	6	7	8	9
Pr(d)	0.301	0.176	0.125	0.097	0.079	0.067	0.058	0.051	0.046

⁴Born Wallace, Nova Scotia, 1835. Professor of Mathematics and Rear Admiral, USN. Died 1909

Emile Borel, in 1909, proved that **almost all real numbers are normal** : in any base, the first digit and all subsequent digits are **uniformly distributed** . For example, with $b = 10$,

$$\Pr(\text{first significant digit} = d) = \frac{1}{9}, \quad d = 1, \dots, 9.$$

However, almost 30 years before, the astronomer **Simon Newcomb**⁴ had proved *Newcomb's Law* : **most natural numbers are logarithmically distributed** :

$$\Pr(\text{first significant digit} = d) = \log_{10}\left(1 + \frac{1}{d}\right), \quad d = 1, \dots, 9,$$

where **Natural Numbers** are those obtained from counting, measurement, or calculation. This gives the following distribution :

d	1	2	3	4	5	6	7	8	9
Pr(d)	0.301	0.176	0.125	0.097	0.079	0.067	0.058	0.051	0.046

⁴Born Wallace, Nova Scotia, 1835. Professor of Mathematics and Rear Admiral, USN. Died 1909

- 1 Generate two sets of random numbers $\{x_i\}, \{y_i\} \in \mathcal{U}(0, 1)$. These will be **normal by design**.⁵
- 2 Form the product $\{z_i = x_i \times y_i\}$.
- 3 Count the frequency of the first significant digit (f.s.d.) of $\{z_i\}$
- 4 Repeat this for 3, 4, \dots , sets of numbers.

Table 1: Distribution p_k for the f.s.d. of a product of k numbers, $k = 2, 3, \infty$.

fsd	1	2	3	4	5	6	7	8	9
p_2	0.301	0.188	0.132	0.099	0.077	0.063	0.053	0.046	0.042
p_3	0.308	0.178	0.124	0.095	0.077	0.065	0.057	0.051	0.046
p_∞	0.301	0.176	0.125	0.097	0.079	0.067	0.058	0.051	0.046

Most computer generated numbers are **Natural**, not Normal.

Applications? In the US the IRS uses it to check fake tax-returns.

⁵Unless you use Randu!

- 1 Generate two sets of random numbers $\{x_i\}, \{y_i\} \in \mathcal{U}(0, 1)$. These will be **normal by design**.⁵
- 2 Form the product $\{z_i = x_i \times y_i\}$.
- 3 Count the frequency of the first significant digit (f.s.d.) of $\{z_i\}$
- 4 Repeat this for 3, 4, ..., sets of numbers.

Table 1: Distribution p_k for the f.s.d. of a product of k numbers, $k = 2, 3, \infty$.

fsd	1	2	3	4	5	6	7	8	9
p_2	0.301	0.188	0.132	0.099	0.077	0.063	0.053	0.046	0.042
p_3	0.308	0.178	0.124	0.095	0.077	0.065	0.057	0.051	0.046
p_∞	0.301	0.176	0.125	0.097	0.079	0.067	0.058	0.051	0.046

Most computer generated numbers are **Natural**, not Normal.

Applications? In the US the IRS uses it to check fake tax-returns.

⁵Unless you use Randu!

- ① Generate two sets of random numbers $\{x_i\}, \{y_i\} \in \mathcal{U}(0, 1)$. These will be **normal by design**.⁵
- ② Form the product $\{z_i = x_i \times y_i\}$.
- ③ Count the frequency of the first significant digit (f.s.d.) of $\{z_i\}$
- ④ Repeat this for 3, 4, \dots , sets of numbers.

Table 1: Distribution p_k for the f.s.d. of a product of k numbers, $k = 2, 3, \infty$.

fsd	1	2	3	4	5	6	7	8	9
p_2	0.301	0.188	0.132	0.099	0.077	0.063	0.053	0.046	0.042
p_3	0.308	0.178	0.124	0.095	0.077	0.065	0.057	0.051	0.046
p_∞	0.301	0.176	0.125	0.097	0.079	0.067	0.058	0.051	0.046

Most computer generated numbers are **Natural**, not Normal.

Applications? In the US the IRS uses it to check fake tax-returns.

⁵Unless you use Randu!

One of the many formulas for **Pythagoras's Constant** π is

$$\frac{\pi}{2} = 2 \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{2n-1}.$$

The sum of the first 50,000 terms gives the approximate value ⁶

```

1.5707 8 632679489 7 619231321 1 9163975 205 209858 3314 6875579 625 874 approx
      |           |           |           |||           ||||           |||
1.5707 9 632679489 6 619231321 6 9163975 144 209858 4699 6875529 104 874 correct
    
```

The vertical bars point from an incorrect digit down to the correct digit of $\pi/2$.

Nico Timme, in his book *Special Functions*, pointed out this phenomenon. The same thing occurs with Euler's series for $\ln 2$.

⁶using PariGP 2.7

One of the many formulas for **Pythagoras's Constant** π is

$$\frac{\pi}{2} = 2 \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{2n-1}.$$

The sum of the first 50,000 terms gives the approximate value ⁶

1.5707 8 632679489 7 619231321 1 9163975 205 209858 3314 6875579 625 874 approx
<div style="display: flex; justify-content: space-around; width: 100%;"> </div>
1.5707 9 632679489 6 619231321 6 9163975 144 209858 4699 6875529 104 874 correct

The vertical bars point from an incorrect digit down to the correct digit of $\pi/2$.

Nico Timme, in his book *Special Functions*, pointed out this phenomenon. The same thing occurs with Euler's series for $\ln 2$.

⁶using PariGP 2.7

Operating Systems

Operating systems can cause strange effects on calculations, and on the people who use them.

Consider the following two well-documented cases . . .

... and then this guy said,

**'Have you ever used
Linux ?'**



