ACM40290: Numerical Algorithms

Floating Point Numbers

Dr Barry Wardell
School of Mathematics and Statistics
University College Dublin

Floating Point Number System

Any real number can be represented by a base b expansion

$$
\begin{aligned}
x &= (d_n d_{n-1} \cdots d_1 d_0 d_{-1} \cdots)_b \\
&= d_n \times b^n + \cdots d_1 \times b + d_0 \times b^0 + d_{-1} \times b^{-1} \cdots \\
&= \sum_{k=-\infty}^{n} d_k b^k
\end{aligned}
$$

Floating Point Number System

All floating point number systems represent a real number x as

$$
\begin{aligned}
fl(x) &= \pm s \times b^e \\
&= \pm(\frac{s_1}{b^1} + \frac{s_2}{b^2} + \cdots + \frac{s_p}{b^p}) \times b^e \\
&= \pm(s_1 s_2 \cdots s_p)_b \times b^e
\end{aligned}
$$

b - base
s - significand (mantissa)
e - exponent
p - precision (number of digits in significand)

Floating Point Number System

Rewrite as

$$
\begin{aligned}
fl(x) &= (s_1 b^{p-1} + s_2 b^{p-2} + \cdots + s_{p-1} b^1 + s_p) \frac{b^e}{b^p} \\
&= \frac{u}{v} \qquad \text{where } u, v \text{ are integers}
\end{aligned}
$$

$\therefore$ A floating point number is a finite rational approximation to x

Floating Point Number System

Each digit in the significand is in the range

$$0 \leq s_i \leq b - 1$$

If $s_1 \neq 0$ then the number is called **normalised**.

The exponent is in the range

$$e_{min} \leq e \leq e_{max}$$

The number 0.0 is represented as

$$fl(0.0) = (0\,0 \cdots 0)_b \times b^0$$

This is **subnormal**.

We denote a particular floating point number system by

$$\mathbb{F}(b, p, e_{min}, e_{max})$$

Floating Point Number System

The range of floating point numbers is given by

$$\cdot \min(s) \times b^{e_{min}} \leq |x| \leq \cdot \max(s) \times b^{e_{max}}$$

If x is normalised

$$\cdot \min(s) = \cdot(1\,0\,0 \cdots 0)_b = b^{-1}$$

$$\cdot \max(s) = \cdot((b-1)(b-1) \cdots (b-1))_b = 1 - b^{-p}$$

$\therefore$ The range of floating point numbers is

$$b^{e_{min}-1} \leq |x| \leq (1 - b^{-p})b^{e_{max}}$$

Distribution of Floating Point Numbers

The distribution of floating point numbers is **Not Uniform**.
For a given exponent, e, there are

$$b^{p-1}(b-1)$$

normalised numbers.

Distribution of Floating Point Numbers

The distribution of floating point numbers is **Not Uniform**.

Example

$$\mathbb{F}(10, 3, 0, 4)$$

Range: $10^{-1} \leq |x| \leq (1 - 10^{-3})10^4$

$$0.1 \leq |x| \leq 9990$$

Normalised numbers for a given exponent: 900

So we have, 900 evenly distributed numbers between

$$0.1 \times 10^e \qquad \text{and} \qquad 0.999 \times 10^e$$

In general, the spacing between numbers is

$$0.001 \times 10^e = 10^{e-3}$$

Spacing for $e = 0, 1, 2, 3, 4$ is 0.001, 0.01, 0.1, 1, 10.

Machine Epsilon

The spacing between any pair of adjacent normalised numbers can be defined in terms of the spacing about 1.0.

Machine epsilon is defined as: the distance between 1.0 and the next highest floating point number.

$$fl(1.0) = \cdot(1\,0\,0\cdots0)_b \times b^1$$

$$fl(1.0 + \epsilon_m) = \cdot(1\,0\,0\cdots01)_b \times b^1$$

$$\therefore \epsilon_m = \cdot(0\,0\,0\cdots01)_b \times b^1 = b^{-p} \times b^1 = b^{1-p}$$

Spacing between 1.0 and the next number **below** is

$$\cdot(1\,0\,0\cdots0)_b \times b^1 - \cdot((b-1)(b-1)\cdots(b-1))_b \times b^0 = b^{-p} = \frac{\epsilon_m}{b}$$

Machine Epsilon

Example

$$\mathbb{F}(10, 3, 0, 4)$$

Machine epsilon is 10^{-2}.
Nearest number above 1.0 is 1.01.
Nearest number below 1.0 is 0.999.

Machine Epsilon

Lemma

The spacing between a normalised floating point number and an adjacent normalised number is at least $b^{-1}\epsilon_m|x|$ and at most $\epsilon_m|x|$

There are a finite number of floating point numbers - there are gaps

Programmers that ignore this often fall into the gaps!

IEEE Floating Point Numbers

Universal standard on modern hardware is IEEE floating point arithmetic (IEEE 754), adopted in 1985

Development led by Prof. William Kahan (Berkeley), received Turing Award in 1989 for this work

	total bits	p	L	U
IEEE Single precision	32	23	-126	127
IEEE Double precision	64	52	-1022	1023

Note that single precision has 8 exponent bits but only 254 different values of E: some exponent values are "reserved"

IEEE Exceptional Values

These exponents are reserved to indicate special behavior, including "exceptional values": Inf, NaN

- Inf $\equiv$ "infinity", e.g. $1/0$ (also $-1/0 = -$Inf)

- NaN $\equiv$ "Not a Number", e.g. $0/0, \mathrm{Inf}/\mathrm{Inf}$

Matlab handles Inf and NaN in a natural way,
e.g. try "`Inf + 1`," "`Inf * 0`" or "`Inf / NaN`"

IEEE Double Precision Numbers

$$\mathbb{F}(2, 53, -1021, 1024)$$

Range: $2^{-1022} \leq |x| < 2^{1024}$

Machine Epsilon: $2^{-52} \approx 2.22\ 10^{-16}$

Unit roundoff: $u = 2^{-53} \approx 1.11\ 10^{-16}$

IEEE Floating-point Numbers

Let $\mathbb{F}$ denote the floating-point numbers, then $\mathbb{F} \subset \mathbb{R}$ and $|\mathbb{F}| < \infty$

Question: How should we represent a real number x which is not in $\mathbb{F}$?

Answer: There are two cases to consider:

- Case 1: x is outside the range of $\mathbb{F}$ (too small or too large)
- Case 2: The mantissa of x requires more than p bits

IEEE Floating-point Numbers

Case 1: x is outside the range of $\mathbb{F}$ (too small or too large)

Too small:

- Smallest positive value that can be represented in double precision[3] is $\approx 10^{-323}$
- For values smaller in magnitude than this we get "Underflow," and value typically gets set to 0

Too large:

- Largest $x \in \mathbb{F}$ ($E = U$ and all mantissa bits are 1) $\approx 2^{1024} \approx 10^{308}$
- For values larger than this we get "Overflow," and value typically gets set to Inf

[3] Smallest value is less than $2^{-1022} \approx 10^{-308}$ because IEEE allows "subnormal" numbers,

IEEE Floating-point Numbers

Case 2: The mantissa of x requires more than p bits

Need to round x to a nearby floating point number

Let `round` : $\mathbb{R} \to \mathbb{F}$ denote our "rounding operator" (several different options: chop, round up, round down, round to nearest)

This introduces a rounding error:

- absolute rounding error: $x - \mathrm{round}(x)$
- relative rounding error: $(x - \mathrm{round}(x))/x$

Floating-point Operations

An arithmetic operation on floating point numbers is called a "floating point operation": $\oplus$, $\ominus$, $\otimes$, $\oslash$ vs. $+, -, \times, /$

Computer performance is often measured in "flops": number of floating point operations per second

Supercomputers are ranked based on number of flops achieved in the "linpack test" which solves dense linear algebra problems

Currently, fastest computers are in the petaflop range:
1 petaflop $= 10^{15}$ floating-point operations per second!

Floating-point Operations

TOP 10 Sites for June 2017

For more information about the sites and systems in the list, click on the links or view the complete list.

1-100 | 101-200 | 201-300 | 301-400 | 401-500

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
5	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890

Floating-point Operations

Modern supercomputers are very large, link many processors together with fast interconnect to minimize communication time

Floating Point Arithmetic

Standard Axioms of algebra **do not apply**.
Floating point arithmetic is **not associative**.
Floating point arithmetic is **not distributive**.

Example

$$\mathbb{F}(10, 3, -5, 5)$$

$$x = 1.00, \quad y = -1.00, \quad z = 0.001$$

$$fl(fl(x + y) + z) = 0.001$$

$$fl(x + fl(y + z)) = 0.000$$

Floating Point Arithmetic

Operations with infinities follow sensible mathematical rules (e.g. finite / inf = 0).

Computing with floating point values may lead to exceptions that may be trapped and halt the program.

Divide by zero: result is $\pm \infty$
Invalid: result is a NaN
Overflow: if result is too large to be represented
Underflow: if result is too small to be represented

Floating Point Arithmetic

Standard Axioms of algebra **do not apply**.
Overflow can cause unexpected behaviour.

Example

$\sqrt{x^2 + y^2}$ may lead to overflow in computing x^2+y^2 even though result does not overflow

In MATLAB, the `hypot` function guards against this

Remedy: $r = \sqrt{x^2 + y^2}$ with $|x| > |y|$

$$= |x|\sqrt{1 + (\tfrac{y}{x})^2}$$

Rounding Error

Let $x = (1.d_1 d_2 \ldots d_p d_{p+1} \ldots)_2 \times 2^E \in \mathbb{R}_{>0}$

Then $x \in [x_-, x_+]$ for $x_-, x_+ \in \mathbb{F}$, where
$x_- = (1.d_1 d_2 \ldots d_p)_2 \times 2^E$ and $x_+ = x_- + \epsilon \times 2^E$

round$(x) = x_-$ or x_+ (depending on rounding rule), hence
$|\text{round}(x) - x| < \epsilon \times 2^E$

Also, $|x| \geq 2^E$

Rounding Error

Therefore we have a relative error of less than ϵ, i.e.:

$$\boxed{\left|\frac{\text{round}(x)-x}{x}\right| < \epsilon}$$

Another standard way to write this is:

$$\text{round}(x) = x\left[1 + \frac{\text{round}(x) - x}{x}\right] = x(1+\delta)$$

where $\delta \equiv \frac{\text{round}(x)-x}{x}$, and $|\delta| < \epsilon$

Hence rounding gives the correct answer to within a factor of $1 + \delta$

Unit Roundoff

The maximum possible value for the relative rounding error is called the **unit roundoff** and is denoted by u. Its value depends on the rounding rule used

$$\mathbf{u} = \begin{cases} b^{1-p} = \epsilon & \text{using chopping,} \\ \frac{1}{2} b^{1-p} = \frac{1}{2}\epsilon & \text{using rounding to nearest} \end{cases}$$

IEEE Floating-point Operation Error

IEEE standard guarantees that for $x, y \in \mathbb{F}$, $x \circledast y = \text{round}(x * y)$ (here $*$ and $\circledast$ represent any one of the 4 arithmetic operations)

Hence from our discussion of rounding error it follows that for $x, y \in \mathbb{F}$, $x \circledast y = (x * y)(1 + \delta)$, for some $|\delta| < \epsilon$

Loss of Precision

Since ϵ is so small, we typically lose very little precision per operation

But loss of precision is not always benign

Errors in Addition and Subtraction

$$
\begin{aligned}
E_{\mathrm{AS}} &= |(x \pm y) - (\tilde{x} \pm \tilde{y})| \\
&= |(x - \tilde{x}) \pm (y - \tilde{y})| \\
&= |(x - x(1+\delta_x)) \pm (y - y(1+\delta_y))| \\
&= |-x\delta_x \pm (-y\delta_y)| \\
&\leq |x\delta_x| + |y\delta_y| \\
&\leq (|x| + |y|)u
\end{aligned}
$$

$$
\boxed{\frac{E_{\mathrm{AS}}}{x \pm y} \leq \frac{|x| + |y|}{|x \pm y|} u}
$$

Catastrophic Cancellation in Subtraction

$$\frac{E_S}{|x-y|} \leq \frac{2u \max\{|x|,|y|\}}{|x-y|}$$

We can get a significant loss of precision if

1. Numbers have same exponent
2. Significant agrees to k digits

$$\mathrm{Rel}(E_S) \approx 2ub^{k+1}$$

Example: Compute the harmonic sum

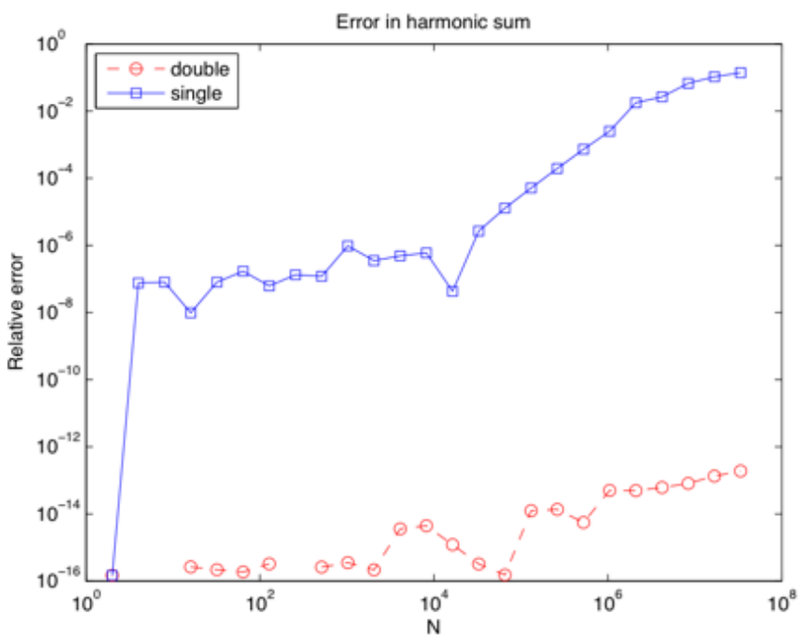
$$H(N) = \sum_{i=1}^{N} \frac{1}{i}$$

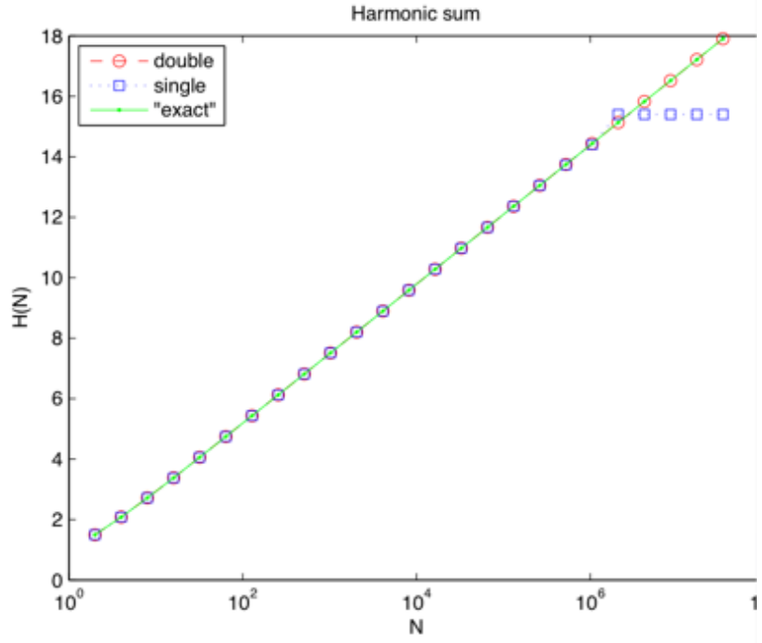
We can do this in forward or reverse order.

```
function nhsum = harmonic(N)
    nhsum = 0.0;
    for i = 1:N   (or  i = N:-1:1)
        nhsum = nhsum + 1.0/i;
    end
end
```

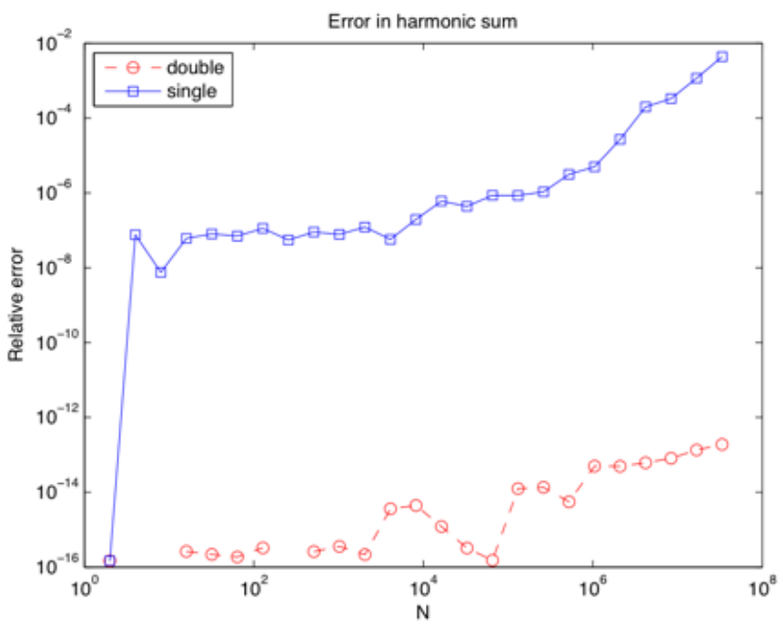
```
% Harmonic number - H(10^4)

% Exact answer to 32 digits
Hn = 1.64483407184805976980 61;

% Forward sum
nsum=0.0;
for i=1:10^4
  nsum = nsum + 1.0/i^2;
end

% Backward sum
nsumr=0.0;
for i=10^4:-1:1
  nsumr = nsumr + 1.0/i^2;
end

[1-nsum/hn, 1-nsumr/hn]
```

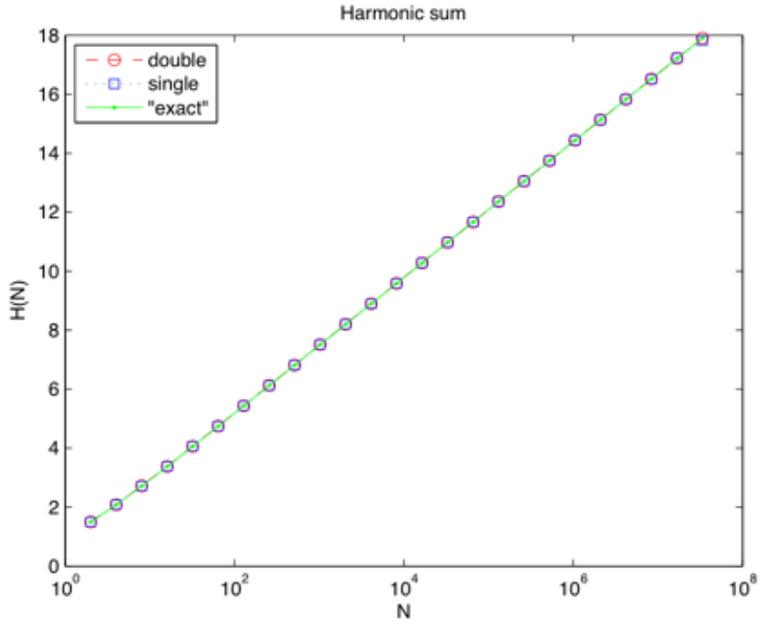
Sum Forward





Sum Back.





Other examples

```
% Roundoff truncation - extreme example
nsum = 0.0;
for i=1:10^8;
    nsum = nsum + 10^-18;
end
nsum = 1.0+nsum;

[nsum,1-nsum/(1.0+10^-10)]
```

```
nsum = 0.0;
nsum = 1.0+nsum;
for i=1:10^8;
    nsum = nsum + 10^-18;
end

[nsum,1-nsum/(1.0+10^-10)]
```

Other examples

```
% Roundoff truncation - less extreme example
nsum = 0.0;
for i=1:10^8;
    nsum = nsum + 10^-12;
end
nsum = 1.0+nsum;

[nsum,1-nsum/(1.0+10^-4)]
```

```
nsum = 0.0;
nsum = 1.0+nsum;
for i=1:10^8;
    nsum = nsum + 10^-12;
end

[nsum,1-nsum/(1.0+10^-4)]
```

Other examples

```
% Roundoff truncation - it doesn't
always work to sum lowest to largest

(-1 + 1) + 10^-18
(1 + 10^-18) - 1
(10^-18 - 1) + 1
```

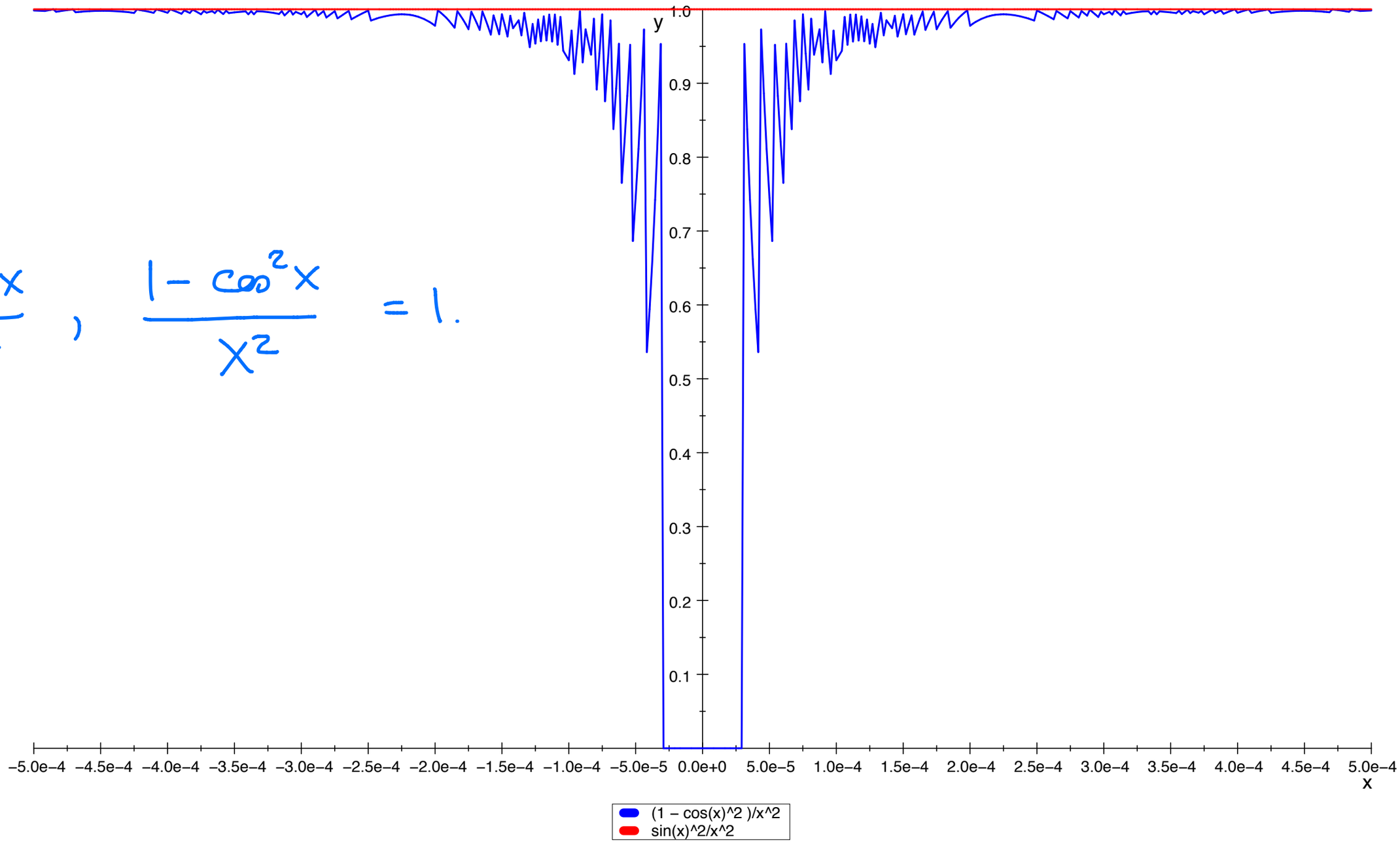
Kahan Summation

Idea: keep track of precision lost in an iteration and try to add it in the next iteration.

```
ksum = 0.0;
ksum = 1.0+ksum;
c = 0.0;
for i = 1:10^8
  y = 10^-12 - c;
  t = ksum + y;
  c = (t - ksum) - y;
  ksum = t;
end
```

$$\frac{\sin^2 x}{x^2} \quad , \quad \frac{1-\cos^2 x}{x^2} = 1.$$

$x \to 0$



Subtraction operator amplifies small error in $\cos^2 x$.

Avoiding Cancellation.

* Rewriting in numerically suitable form

$\sqrt{x+\delta} - \sqrt{x}$ Use $\dfrac{\delta}{\sqrt{x+\delta} + \sqrt{x}}$

$x^2 - y^2$ Use $(x-y)(x+y)$.

* Taylor Expansions.

$\sqrt{x+\delta} - \sqrt{x} \approx \dfrac{\delta}{2\sqrt{x}}$ for $\delta \ll x$

Error of multiplication and division

$$E_m = |(X \otimes y) - (\hat{X} \otimes \hat{y})|$$

$$= |X \otimes y - X(1+\delta_x) \otimes y(1+\delta_y)|$$

$$= |(X \otimes y)(1 - (1+\delta_x)(1+\delta_y))|$$

$$\leq |X \otimes y||\delta_x + \delta_y| \leq |X \otimes y| 2u$$

$$Rel(E_m) = \frac{E_m}{|X \otimes y|} \leq 2u$$

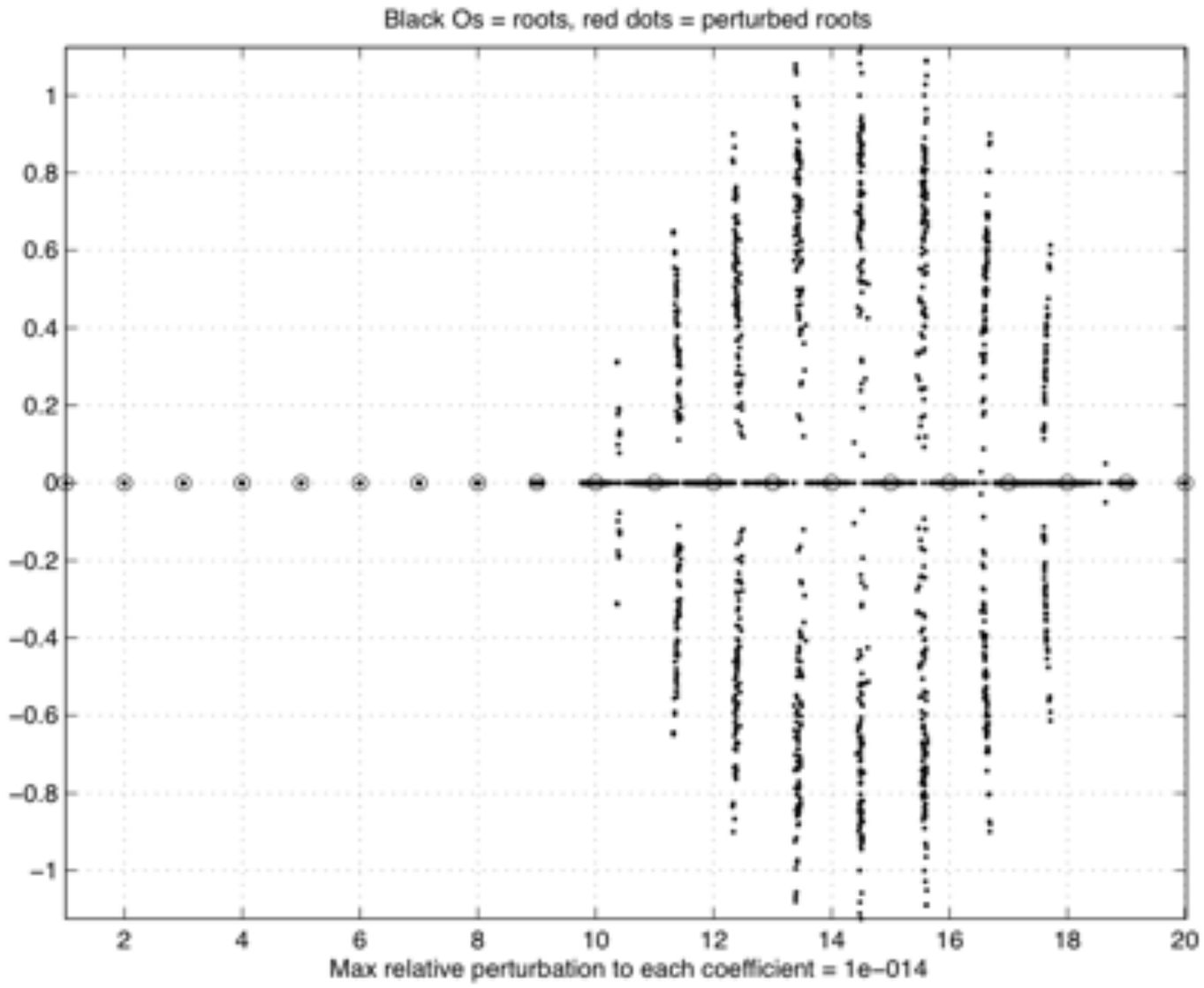
Relative Error is independent of x and y.

Wilkinson's Polynomial

$$P_{20}(x) = \prod_{i=1}^{20} (x - i)$$

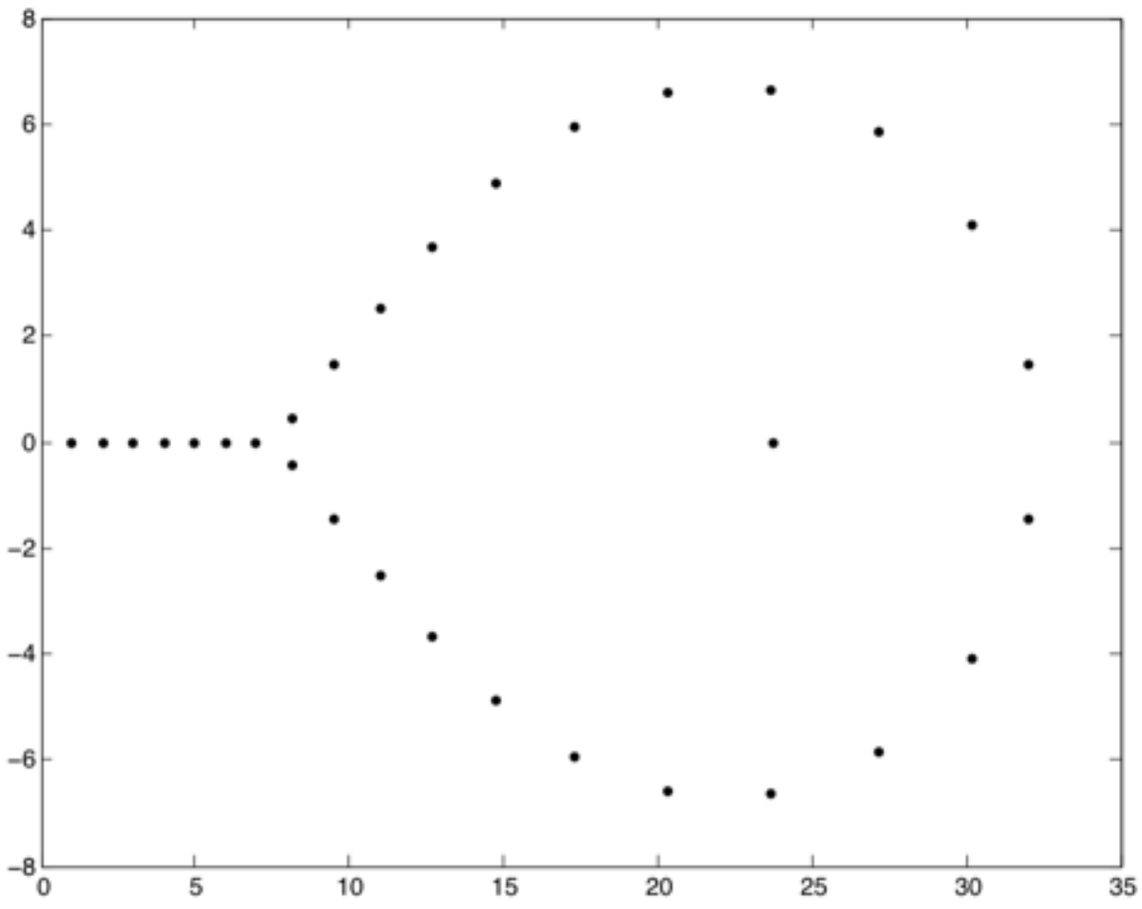
$$= (x-1)(x-2)\ldots(x-20)$$

$$= 20! + \cdots + 210x^{19} + x^{20}$$



Coefficients of the original polynomial randomly perturbed by a small amount.

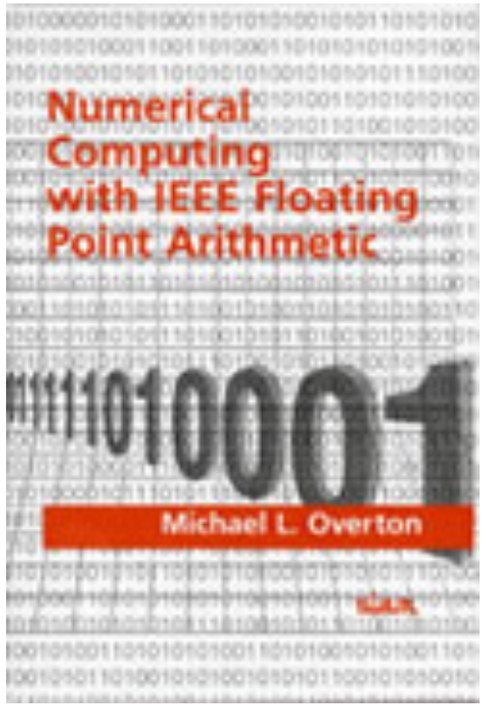
No numerical method can compensate for an ill-conditioned problem.




```
pw30 = poly(1:30) ;
zw30 = roots(pw30) ;
plot(zw30, '.k')
```

IEEE Floating Point Arithmetic

For more detailed discussion of floating point arithmetic, see:



"Numerical Computing with IEEE Floating Point Arithmetic,"
Michael L. Overton, SIAM, 2001