Errors on Computation: FP Numbers

Overflow, Underflow, EPS de la maquina, etc.

William Oquendo

Credits: ICTP (Stefano Cozzini); Computational Physics - Landau and Paez; Sirca - Computational Methods

Bits, Bytes, ...

- BIT: 1 o 0, true or false, yes or no

0	0
1	1
01	1
10	2
11	3
1101	???

  Solo hay 10 tipos de personas: Las que entienden binario y las que no.

- If we are given N bits, the first one is used for the sign, and the remainder for representing the number: 2^{N-1}

- BYTE: 8 bits, 256 values

- 2^10 = 1,024 = 1 Kbyte (small difference with 1000 !!!)

Representing numbers

- Real numbers = Impossible (unlimited accuracy)
- In a computer: Binary Coded Decimal, Rational Numbers (Mathematica), Fixed Point, Floating Point.
- IEEE 754 standard for floating number : operations, rounding, etc.

FP Number representation

- Fixed notation:

$$x_{\text{fix}} = \text{sign} \times (\alpha_n 2^n + \alpha_{n-1} 2^{n-1} + \cdots + \alpha_0 2^0 + \cdots + \alpha_{-m} 2^{-m}).$$

Advantage: All numbers have the same error, 2^{-m-1}
Disadvantage: Small numbers have LARGE relative errors
Applications: Bussiness.

- Floating point notation: (Scientific notation!)

$$x_{\text{float}} = (-1)^s \times \text{mantissa} \times 2^{\text{expfld - bias}}.$$

Example: Single precision 32 bits number, 8 bits fot the exponent [0, 255], negative exponents are represented with bias equal to -127. Of the remainder bits, one is used for the sign and the others for the mantissa

Number representation

- Floating (continued ...):
  Single precision (4 byte): 6-7 decimal places of precision, 1 part in 2^23, 10^{-44} < single precision < 10^{38}

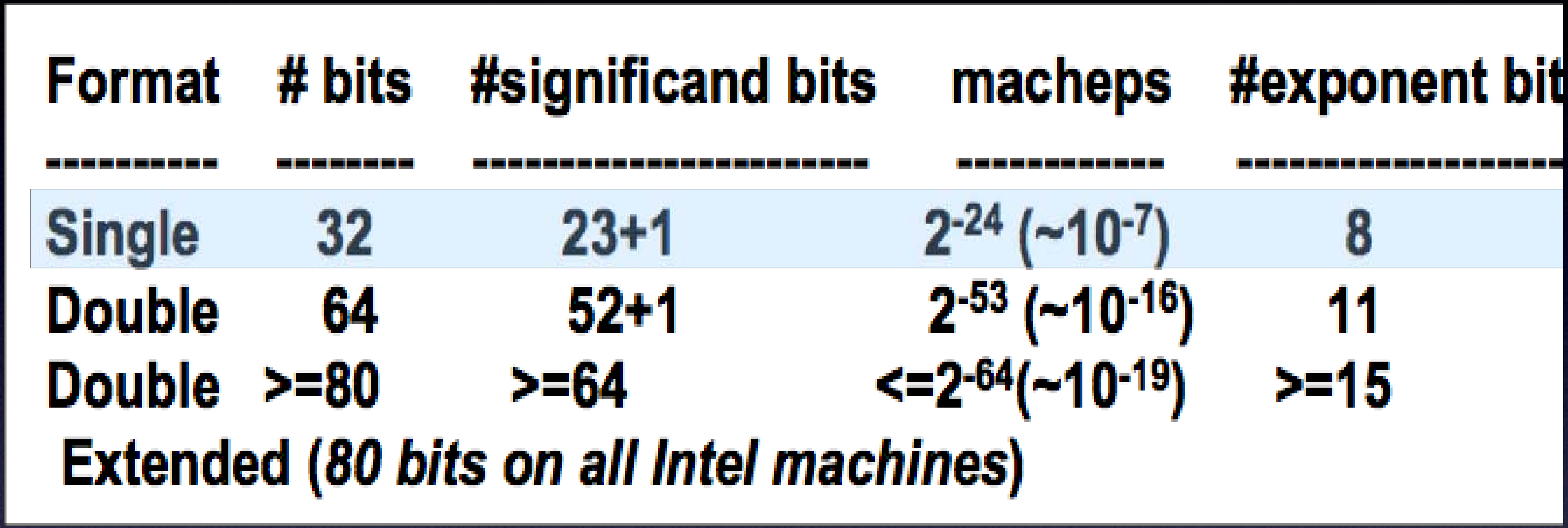
- Mantissa:

$$\text{mantissa} = m_1 \times 2^{-1} + m_2 \times 2^{-2} + \cdots + m_{23} \times 2^{-23},$$

- Double precision: 64 bits (8 bytes), 11 bits for the exponent, and 52 for the mantissa, 16 decimal places of precision (1 in 2^52), and range in

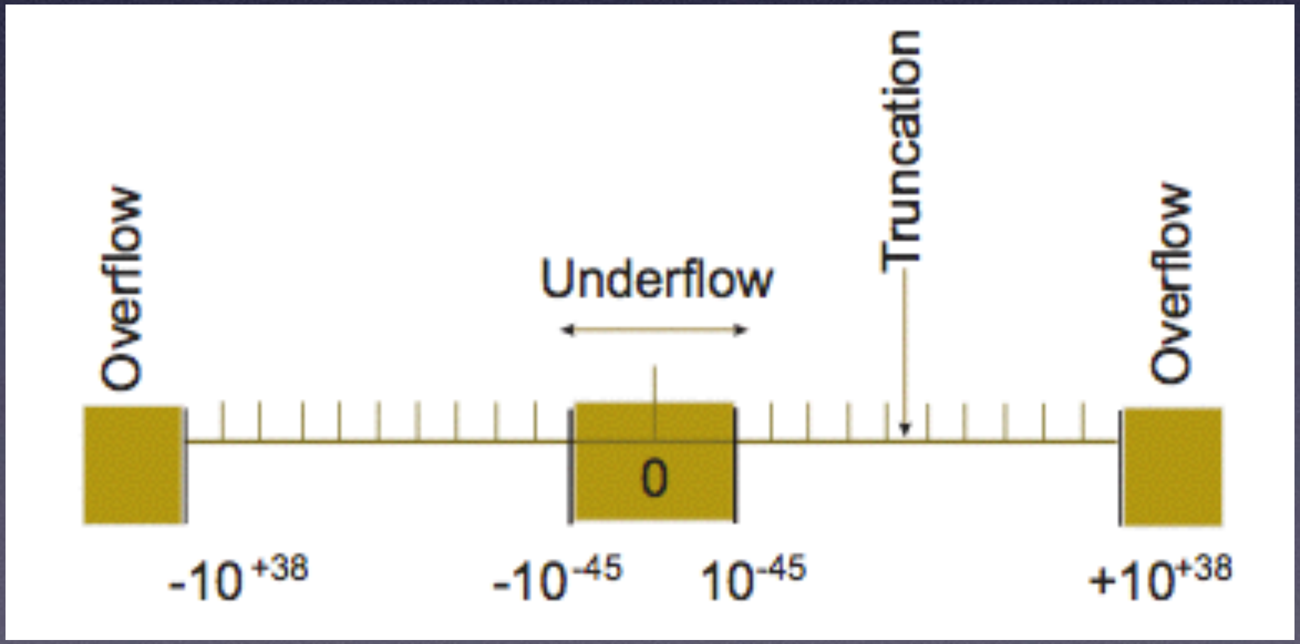
$$10^{-322} \leq \text{double precision} \leq 10^{308}$$

FP Number Range

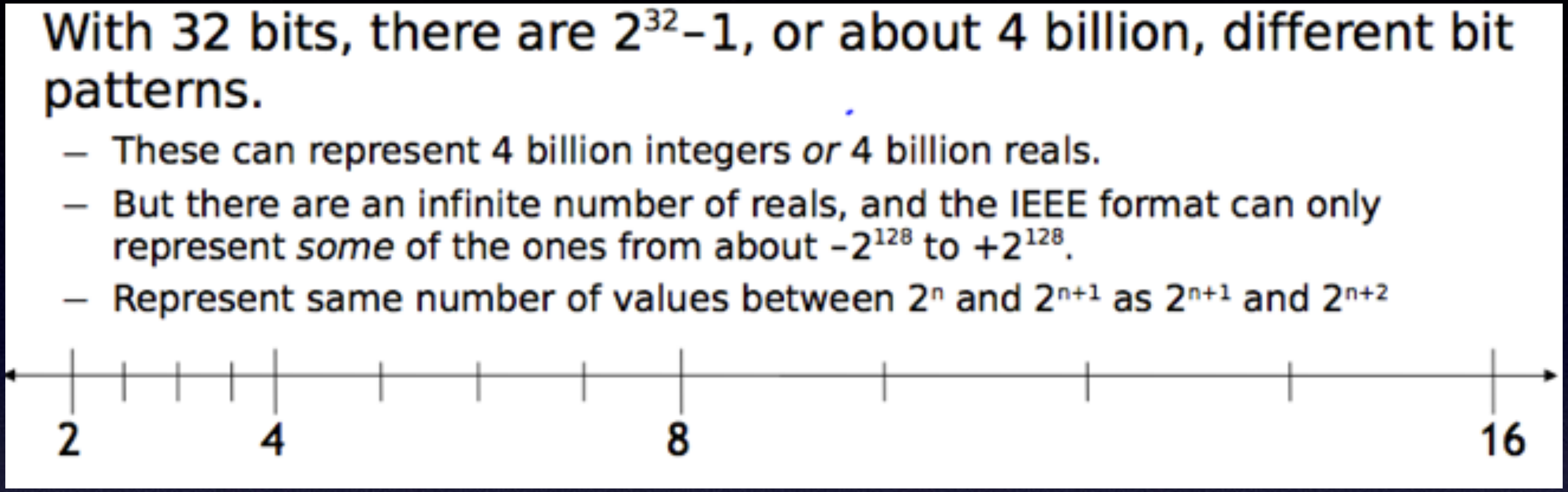


Format	# bits	#significand bits	macheps	#exponent bit
----------	--------	-----------------------	------------	-------------------
Single	32	23+1	2^{-24} (~10^{-7})	8
Double	64	52+1	2^{-53} (~10^{-16})	11
Double	>=80	>=64	<=2^{-64}(~10^{-19})	>=15

Extended (*80 bits on all Intel machines*)



FP Number Density



With 32 bits, there are $2^{32}-1$, or about 4 billion, different bit patterns.

- These can represent 4 billion integers *or* 4 billion reals.
- But there are an infinite number of reals, and the IEEE format can only represent *some* of the ones from about -2^{128} to $+2^{128}$.
- Represent same number of values between 2^n and 2^{n+1} as 2^{n+1} and 2^{n+2}

- Same number of bits to represent all numbers : the smaller the number, the greater the density of representable numbers.
- Example: There are 8388607 single precision numbers between 1.0 and 2.0, while there are only about 8191 between 1023.0 and 1024.0
- The larger the number, the smaller the number of fp numbers to use, then the larger the truncation error.

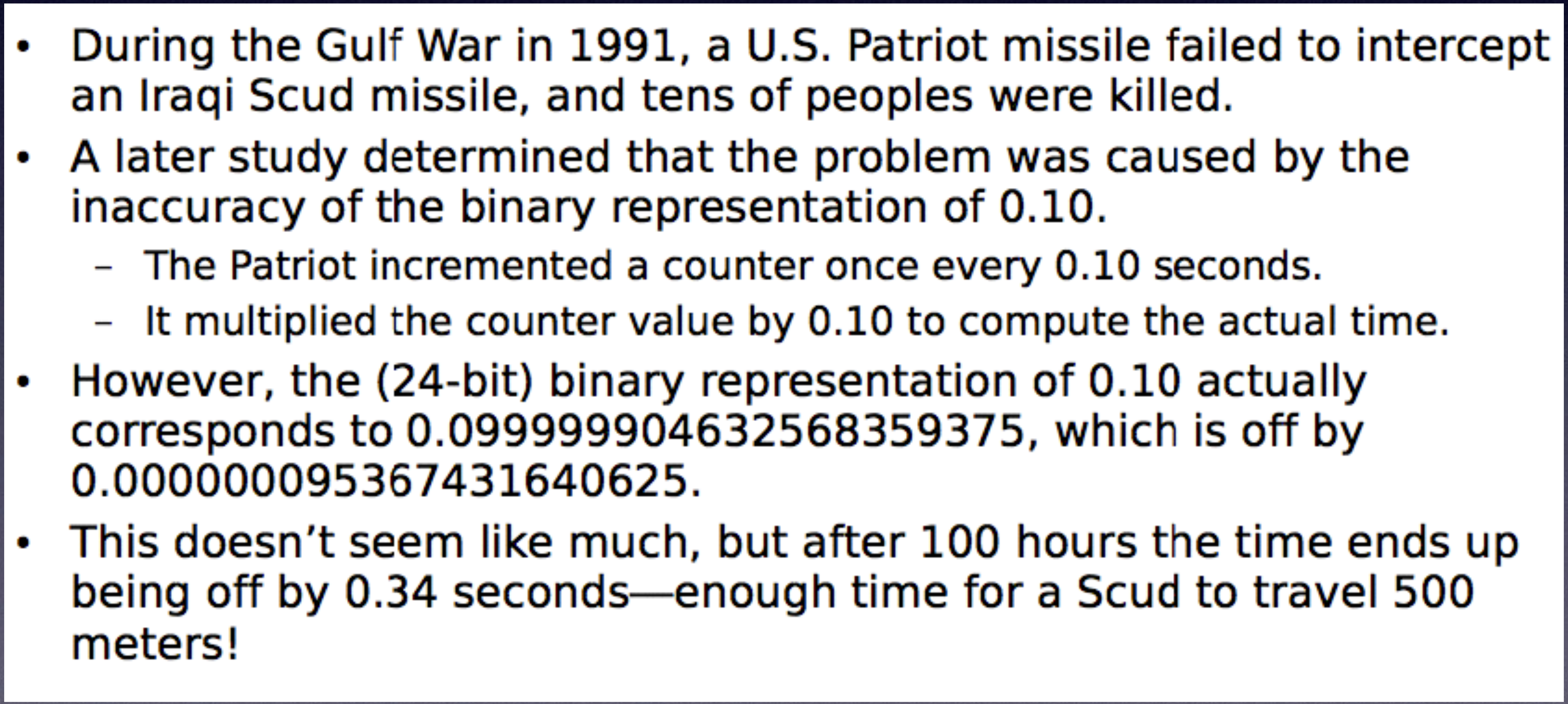
Some issues : ieee 754 helps

- Not all mathematical properties are preserved (no commutativity)
- What to do for 0/0, overflow, underflow, truncation, etc?
- Binary/Decimal conversion.
- IEEE helps for portability, but should be implemented at langauge level.
- Fortran: REAL*4/REAL (32 bit); REAL*8/DOUBLE PRECISION (64 bits); REAL*16 (Not always present)
- C/C++: float (32 bits); double (64 bits); long double (80 bit)

Misconceptions

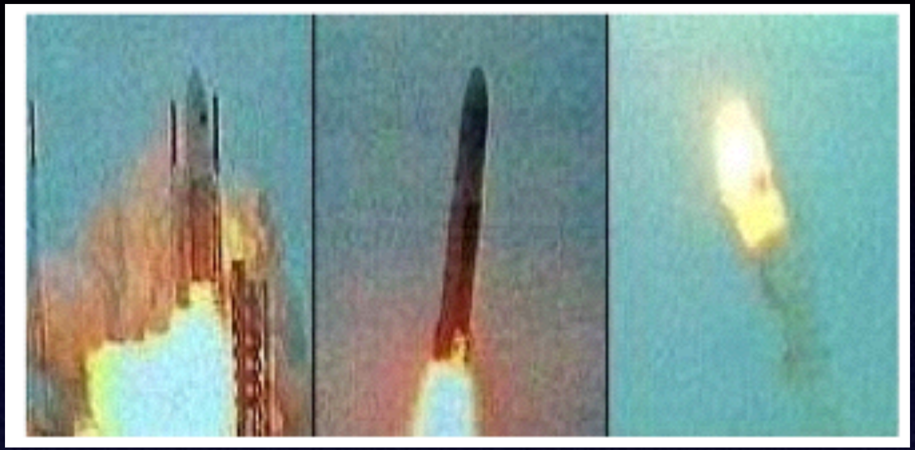
- FP arithmetic is not well defined: false, IEEE 754 standarizes.
- 15 decimal digits are enough for my simple 3-decimal digits calculation: false, 14 significant digits can be destroyed in a single operation.
- I can always cast a float to integer: be careful!
- Addition is associative: false! x + (y + z) != (x + y) + z , with x = -1.5e38, y = 1.5e38, z = 1.
- Everything can be represented: 1/3 is not, 0.01 is not, 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 != 1.0
- The compiler takes care: NO! -ffloat-store (and related) for gnu compilers; -mp for intel compiler; and -Kieee for

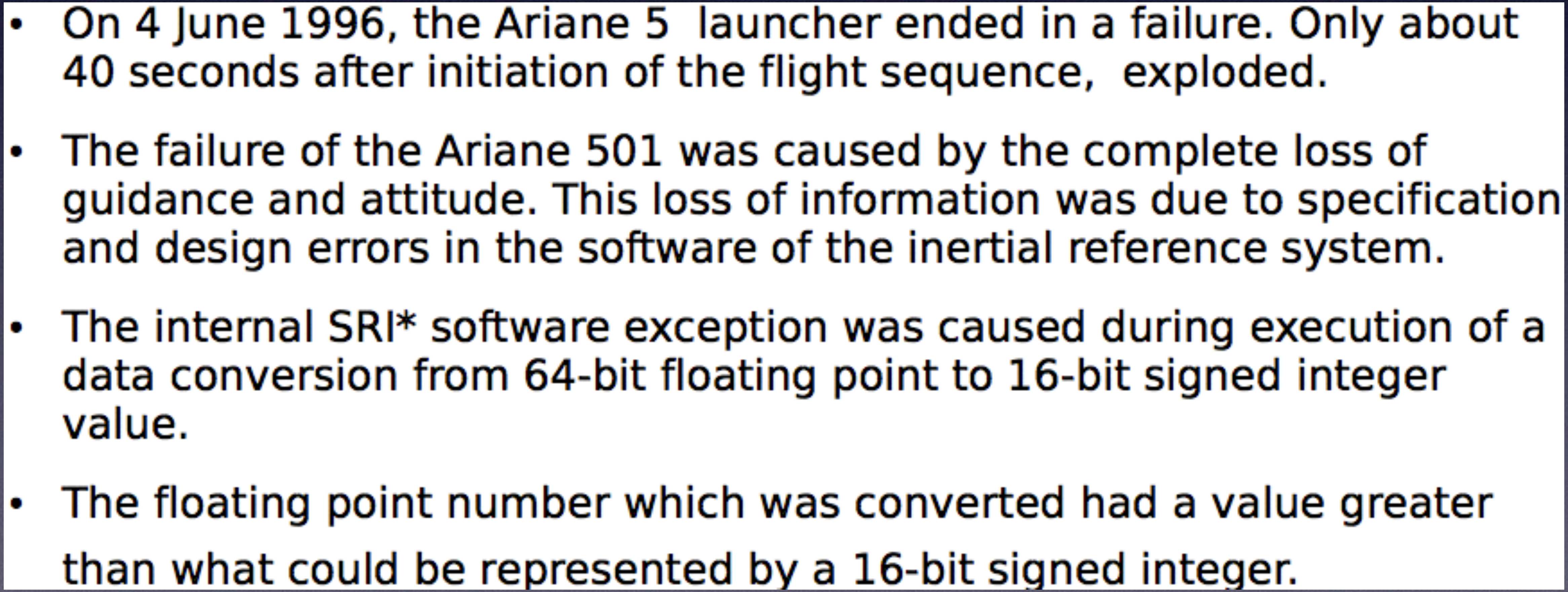
Affects real-life



- During the Gulf War in 1991, a U.S. Patriot missile failed to intercept an Iraqi Scud missile, and tens of peoples were killed.
- A later study determined that the problem was caused by the inaccuracy of the binary representation of 0.10.
  - The Patriot incremented a counter once every 0.10 seconds.
  - It multiplied the counter value by 0.10 to compute the actual time.
- However, the (24-bit) binary representation of 0.10 actually corresponds to 0.099999904632568359375, which is off by 0.000000095367431640625.
- This doesn't seem like much, but after 100 hours the time ends up being off by 0.34 seconds—enough time for a Scud to travel 500 meters!

Affects real-life

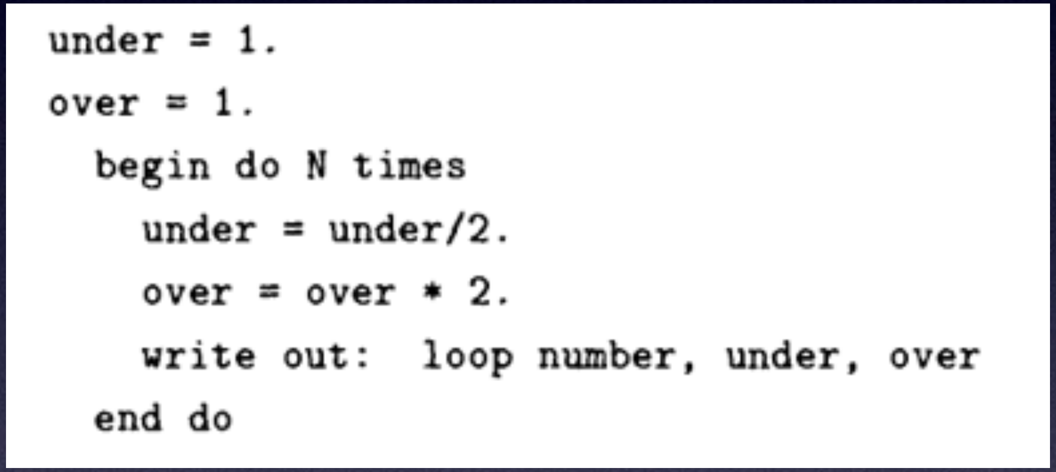




- On 4 June 1996, the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, exploded.
- The failure of the Ariane 501 was caused by the complete loss of guidance and attitude. This loss of information was due to specification and design errors in the software of the inertial reference system.
- The internal SRI* software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value.
- The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.

Overflow and underflow

- Overflow: Represent a number larger than the largest number the computer is able to represent.



```
under = 1.
over = 1.
  begin do N times
    under = under/2.
    over = over * 2.
    write out:  loop number, under, over
  end do
```

- Underflow: Represent a number smaller than the smallest number the computer is able to represent.
- Exercise: Implement overflow, underflow for float and double.
- Exercise: the same for integers, add and substract one.

Machine precision

- How precise is the computer number representation? Assume single precision.

$$7 \quad + \quad 1.0 \times 10^{-7} =?$$

$$\begin{aligned}
7 &= 0 \;\; 10000010 \;\; 1110\;0000\;0000\;0000\;0000\;000, \\
10^{-7} &= 0 \;\; 01100000 \;\; 1101\;0110\;1011\;1111\;1001\;010,
\end{aligned}$$

$$\begin{aligned}
10^{-7} &= 0 \;\; 01100001 \;\; 0110\;1011\;0101\;1111\;1100101\;(0) \\
&= 0 \;\; 01100010 \;\; 0011\;0101\;1010\;1111\;1110010\;(10) && (2.13) \\
\cdots &, \\
&= 0 \;\; 10000010 \;\; 0000\;0000\;0000\;0000\;0000\;000\;(0001101\cdots) \\
\Rightarrow & \;\; 7 \quad + \quad 1.0 \times 10^{-7} = 7 && (2.14)
\end{aligned}$$

Machine precision

- Machine EPS:

$$1_c + \epsilon_m = 1_c,$$

$$x_c = x(1+\epsilon), \quad |\epsilon| \le \epsilon_m \ .$$

- Exercise: Machine precision computation for float and double

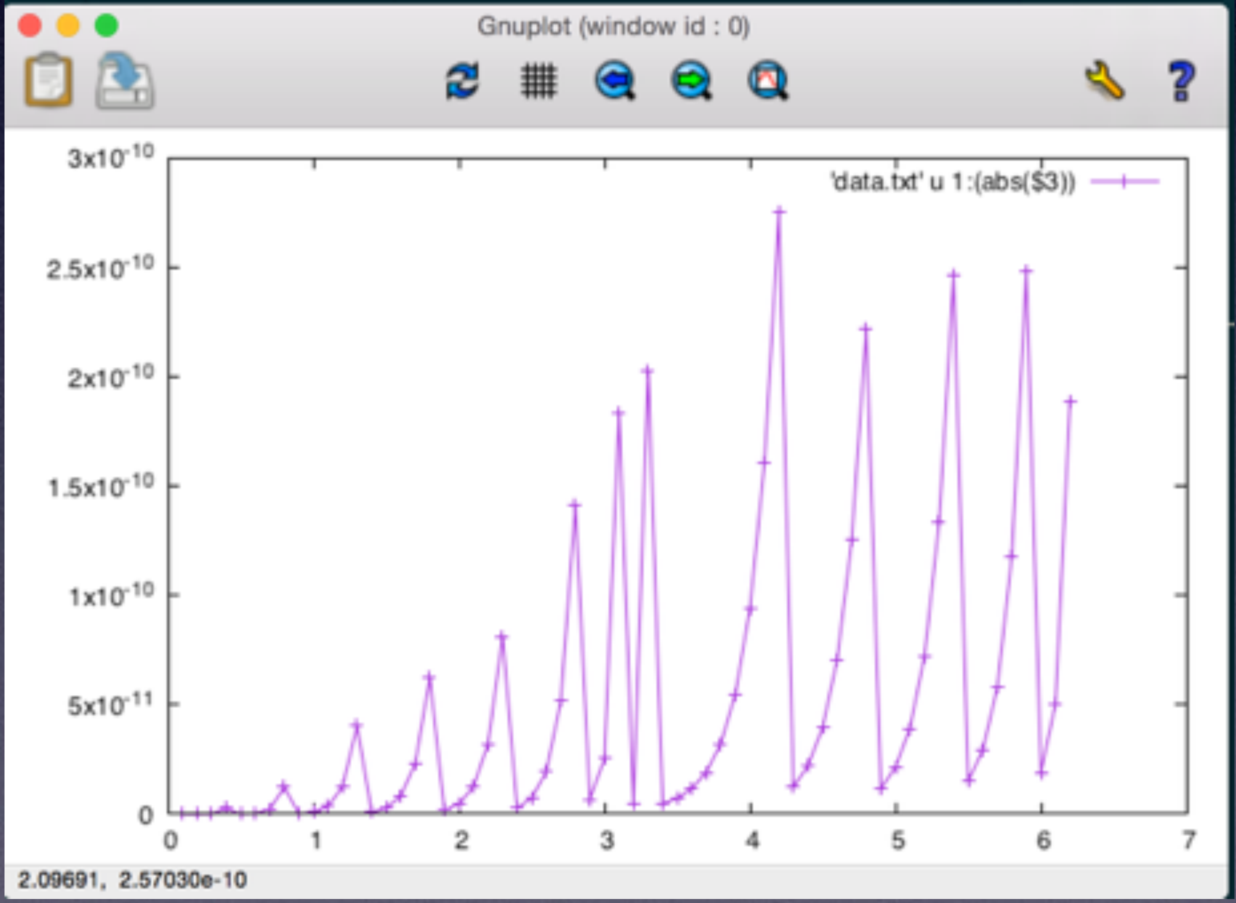
```
eps = 1.
  begin do N times
    eps = eps/2.
    one = 1.  + eps
    write out:  loop number, one, eps
  end do
```

Exercise

Your *problem* is to use just this series to calculate $\sin x$ for $x < 2\pi$ and $x > 2\pi$, with an absolute error in each case of less than 1 part in 10^8. While an infinite series

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \quad \text{(exact)} .$$

$$\sin x \simeq \sum_{n=1}^{N} \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!} \quad \text{(algorithm)} .$$



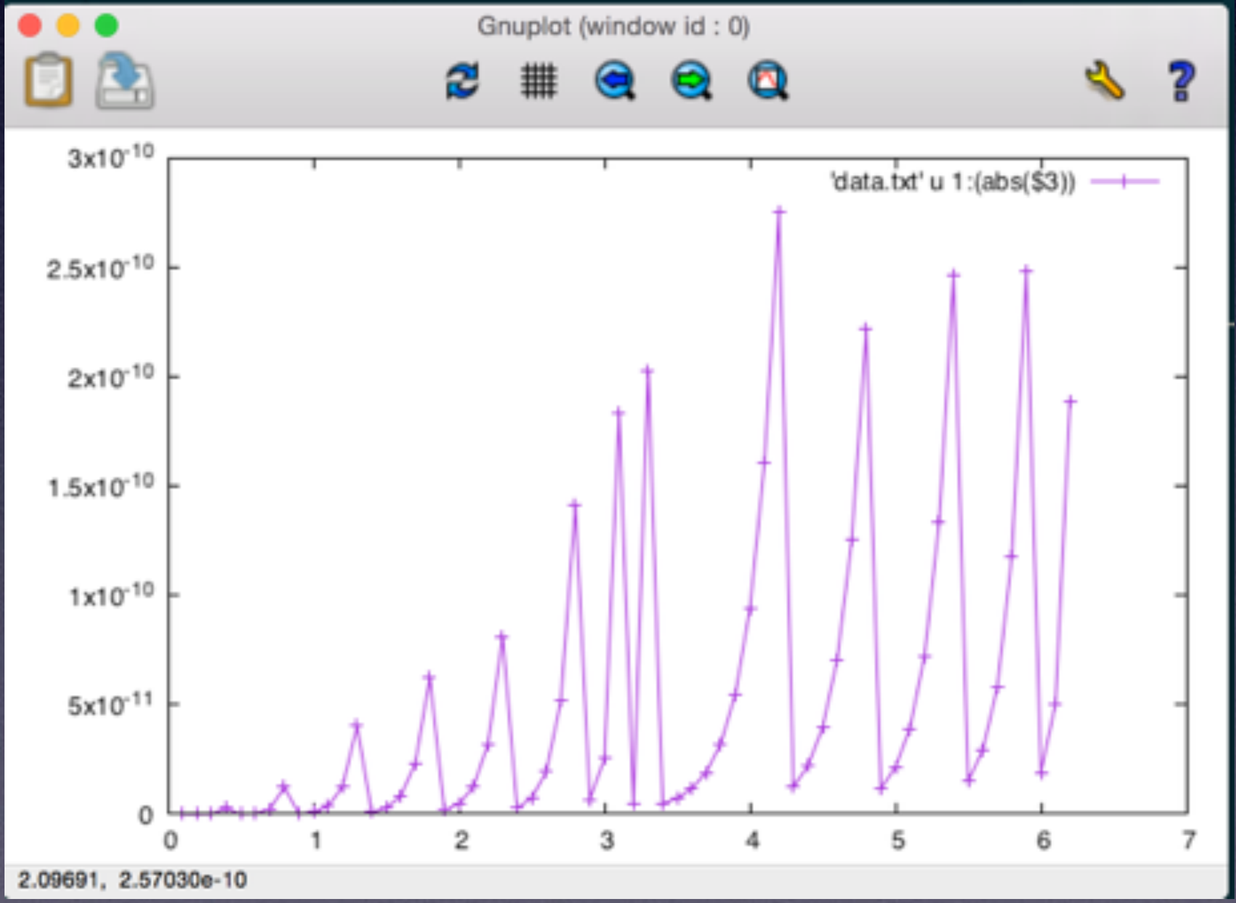
Exercise

Your *problem* is to use just this series to calculate $\sin x$ for $x < 2\pi$ and $x > 2\pi$, with an absolute error in each case of less than 1 part in 10^8. While an infinite series

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \quad \text{(exact)}\,.$$

$$\sin x \simeq \sum_{n=1}^{N} \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!} \quad \text{(algorithm)}\,.$$

1. Write a program that implements this pseudocode for the indicated x values. Present the results as a table with headings `x imax sum |sum- sin(x)|/sin(x)`, where `sin(x)` is the value obtained from the built-in function. The last column here is the relative error in your computation. Modify the code that sums the series in a "good way" (no factorials) to one that calculates the sum in a "bad way" (explicit factorials).
2. Produce a table as above.
3. Start with a tolerance of 10^{-8} as in (2.29).
4. Show that for sufficiently small values of x, your algorithm converges (the changes are smaller than your tolerance level) and that it converges to the correct answer.
5. Compare the number of decimal places of precision obtained with that expected from (2.29).
6. Without using the identity $\sin(x + 2n\pi) = \sin(x)$, show that there is a range of somewhat large values of x for which the algorithm converges, but that it converges to the wrong answer.
7. Show that as you keep increasing x, you will reach a regime where the algorithm does not even converge.
8. Now make use of the identity $\sin(x + 2n\pi) = \sin(x)$ to compute $\sin x$ for large x values where the series otherwise would diverge.
9. Repeat the calculation using the "bad" version of the algorithm (the one that calculates factorials) and compare the answers.
10. Set your tolerance level to a number smaller than machine precision and see how this affects your conclusions.



Summing series

- Exercise:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots \quad (x^2 < \infty).$$

```
term = 1, sum = 1, eps = 10**(-8)                              Initialize.
do
  term = -term * x/i                                New term in terms of old.
  sum = sum + term                                               Add in term.
  while abs(term/sum) > eps                    Break iteration if accurate.
end do
```

Kind of errors

- Prob. of an error:

$$\text{start} \to U_1 \to U_2 \to \ldots \to U_n \to \text{end},$$

- Blunders/Theoretical: Typographical, wrong program, etc
- Random errors: Electronics, martial invasion, etc.
- Approximation: (mathematical series truncation)
- Roundoff: Truncation of a number in the computer representation

Kind of errors

- Prob. of an error:

$$\text{start} \to U_1 \to U_2 \to \ldots \to U_n \to \text{end},$$

- Blunders/Theoretical: Typographical, wrong program, etc
- Random errors: Electronics, martial invasion, etc.
- Approximation: (mathematical series truncation)
- Roundoff: Truncation of a number in the computer representation

$$2\left(\frac{1}{3}\right) - \frac{2}{3} = 0.6666 - 0.6667 = -0.0001 \neq 0\ .$$

Kind of errors

- Prob. of an error:

$$\text{start} \rightarrow U_1 \rightarrow U_2 \rightarrow \ldots \rightarrow U_n \rightarrow \text{end},$$

- Blunders/Theoretical: Typographical, wrong program, etc
- Random errors: Electronics, martial invasion, etc.
- Approximation: (mathematical series truncation)
- Roundoff: Truncation of a number in the computer representation

$$2\left(\frac{1}{3}\right) - \frac{2}{3} = 0.6666 - 0.6667 = -0.0001 \neq 0 .$$

$$a = 11\,223\,344\,556\,677\,889\,900 = 1.122\,334\,455\,667\,788\,99 \times 10^{19} .$$

Substractive cancellation

$$\begin{aligned} a = b - c \quad \Rightarrow \quad a_c &= b_c - c_c, \\ a_c &= b(1+\epsilon_b) - c(1+\epsilon_c), \\ \Rightarrow \quad \frac{a_c}{a} &= 1 + \epsilon_b \frac{b}{a} - \frac{c}{a}\epsilon_c. \end{aligned}$$

$$\begin{aligned} \frac{a_c}{a} &= 1 + \epsilon_a, \\ \epsilon_a &\simeq \frac{b}{a}(\epsilon_b - \epsilon_c). \end{aligned}$$

Substractive cancellation: Example

$$ax^2 + bx + c = 0$$

Substractive cancellation: Example

$$ax^2 + bx + c = 0$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{or} \quad x'_{1,2} = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}} \; .$$

Substractive cancellation: Example

$$ax^2 + bx + c = 0$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{or} \quad x'_{1,2} = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}} .$$

- Escribir un programa que calcula las soluciones para cualquier a, b, c.
- Explorar las diferencias para a=1, b=1, c=1e-n, con n=1,2,3,…
- Seleccionar la forma más precisa.

Substractive cancellation: Example

$$S_N^{(1)} = \sum_{n=1}^{2N} (-1)^n \frac{n}{n+1}.$$

$$S_N^{(2)} = -\sum_{n=1}^{N} \frac{2n-1}{2n} + \sum_{n=1}^{N} \frac{2n}{2n+1}.$$

$$S_N^{(3)} = \sum_{n=1}^{N} \frac{1}{2n(2n+1)}.$$

- Escribir un programa que calcula cada suma como funcion de N.
- Suponer que S3 es exacta. Hacer una tabla que compare a S1 y a S2 con S3 de forma relativa: (S1 - S3)/S3 en funcion de N
- Dibujar y analizar.

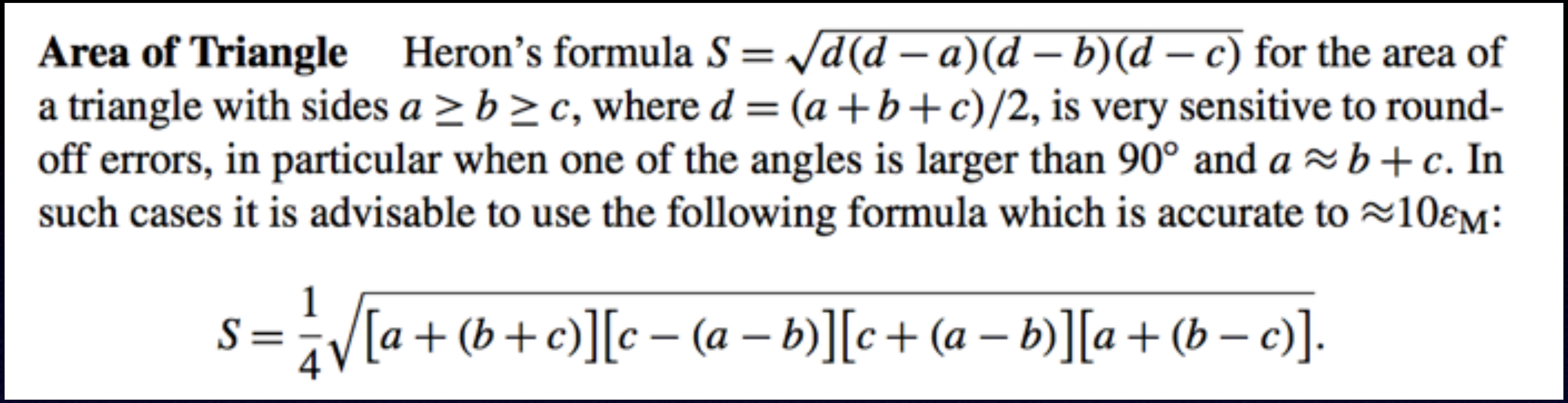
Substractive cancellation: Example

$$S^{(\mathrm{up})} = \sum_{n=1}^{N} \frac{1}{n},$$

$$S^{(\mathrm{down})} = \sum_{n=N}^{1} \frac{1}{n}.$$

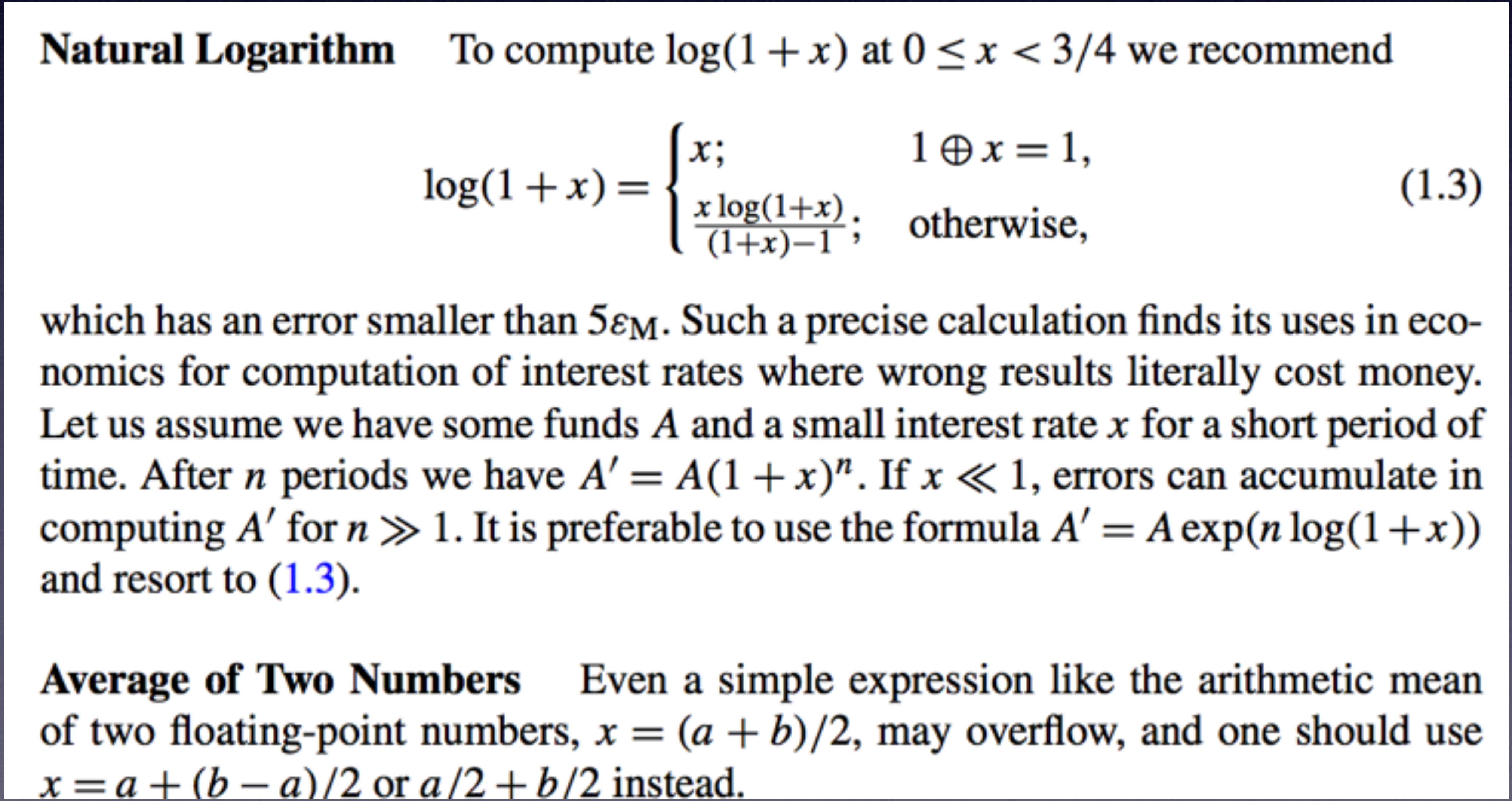
- Escribir un programa que calcula cada suma como funcion de N.
- Hacer una tabla de la diferencia relativa dividida entre la suma relativa como funcion de N.
- Dibujar y analizar.

Substractive cancellation: Example



Area of Triangle Heron's formula $S = \sqrt{d(d-a)(d-b)(d-c)}$ for the area of a triangle with sides $a \geq b \geq c$, where $d = (a+b+c)/2$, is very sensitive to round-off errors, in particular when one of the angles is larger than 90° and $a \approx b + c$. In such cases it is advisable to use the following formula which is accurate to $\approx 10\varepsilon_M$:

$$S = \frac{1}{4}\sqrt{[a+(b+c)][c-(a-b)][c+(a-b)][a+(b-c)]}.$$



Natural Logarithm To compute $\log(1+x)$ at $0 \leq x < 3/4$ we recommend

$$\log(1+x) = \begin{cases} x; & 1 \oplus x = 1, \\ \frac{x\log(1+x)}{(1+x)-1}; & \text{otherwise}, \end{cases} \qquad (1.3)$$

which has an error smaller than $5\varepsilon_M$. Such a precise calculation finds its uses in economics for computation of interest rates where wrong results literally cost money. Let us assume we have some funds A and a small interest rate x for a short period of time. After n periods we have $A' = A(1+x)^n$. If $x \ll 1$, errors can accumulate in computing A' for $n \gg 1$. It is preferable to use the formula $A' = A\exp(n\log(1+x))$ and resort to (1.3).

Average of Two Numbers Even a simple expression like the arithmetic mean of two floating-point numbers, $x = (a+b)/2$, may overflow, and one should use $x = a + (b-a)/2$ or $a/2 + b/2$ instead.

Multiplicative errors

$$
\begin{aligned}
a = b \times c \quad &\Rightarrow \quad a_c = b_c \times c_c, \\
&\Rightarrow \quad \frac{a_c}{a} = \frac{(1+\epsilon_b)(1+\epsilon_c)}{(1+\epsilon_a)} \simeq 1 + \epsilon_b + \epsilon_c.
\end{aligned}
$$

$$R \approx \sqrt{N} r.$$

$$\epsilon_{\mathrm{ro}} \approx \sqrt{N} \epsilon_m.$$

- Sometimes errors are not random but coherent!!!

Errors in algorithms

$$\epsilon_{\text{aprx}} \simeq \frac{\alpha}{N^{\beta}}.$$

$$\epsilon_{\text{ro}} \simeq \sqrt{N}\epsilon_m,$$

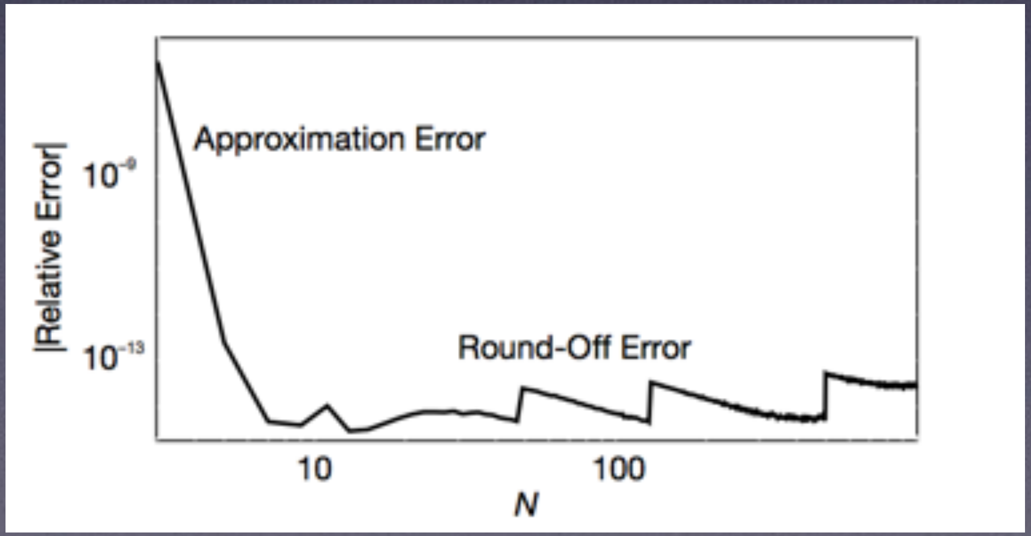
$$\begin{aligned}\epsilon_{\text{tot}} &= \epsilon_{\text{aprx}} + \epsilon_{\text{ro}},\\ &\simeq \frac{\alpha}{N^{\beta}} + \sqrt{N}\epsilon_m.\end{aligned}$$

Errors in algorithms

$$\epsilon_{\text{aprx}} \simeq \frac{\alpha}{N^{\beta}}. \quad \longleftrightarrow \quad \epsilon_{\text{ro}} \simeq \sqrt{N}\epsilon_m,$$

$$\updownarrow$$

$$\begin{aligned} \epsilon_{\text{tot}} &= \epsilon_{\text{aprx}} + \epsilon_{\text{ro}}, \\ &\simeq \frac{\alpha}{N^{\beta}} + \sqrt{N}\epsilon_m. \end{aligned}$$



Errors in algorithms

$$\begin{aligned}\epsilon_{\text{tot}} &= \epsilon_{\text{aprx}} + \epsilon_{\text{ro}}, \\ &\simeq \frac{\alpha}{N^{\beta}} + \sqrt{N}\epsilon_m .\end{aligned}$$

$$\frac{d\epsilon_{tot}}{dN} = -\frac{\alpha\beta}{N^{\beta+1}} + \frac{\epsilon_m}{2\sqrt{N}} = 0$$

$$N^* = \left(\frac{2\alpha\beta}{\epsilon_m}\right)^{\frac{2}{2\beta+1}}$$

Errors in algorithms

$$N^* = \left(\frac{2\alpha\beta}{\epsilon_m}\right)^{\frac{2}{2\beta+1}}$$

β	N float $\epsilon_m = 10^{-6}$	N double $\epsilon_m = 10^{-16}$	float ϵ_{min}	double ϵ_{min}
2	437	4373448	2.6E-05	2.68E-15
4	34	5705	6.7E-06	8.5E-15

$\alpha = 1$

Numerical Stability

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \longrightarrow u_j^{n+1} = u_j^n + r\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right) \qquad r = \frac{\alpha \Delta t}{\Delta x^2}$$

Define the round-off error ϵ_j^n as

$$\epsilon_j^n = N_j^n - u_j^n \longrightarrow \epsilon_j^{n+1} = \epsilon_j^n + r\left(\epsilon_{j+1}^n - 2\epsilon_j^n + \epsilon_{j-1}^n\right)$$

$$\epsilon(x) = \sum_{m=1}^{M} A_m e^{ik_m x} \longrightarrow \epsilon(x,t) = \sum_{m=1}^{M} e^{at} e^{ik_m x} \longrightarrow \epsilon_m(x,t) = e^{at} e^{ik_m x}$$

$$e^{a\Delta t} = 1 - \frac{4\alpha \Delta t}{\Delta x^2} \sin^2(k_m \Delta x/2) \longrightarrow G \equiv \frac{\epsilon_j^{n+1}}{\epsilon_j^n} \longrightarrow G = \frac{e^{a(t+\Delta t)} e^{ik_m x}}{e^{at} e^{ik_m x}} = e^{a\Delta t}$$

$$\left|1 - \frac{4\alpha \Delta t}{\Delta x^2} \sin^2(k_m \Delta x/2)\right| \leq 1 \longrightarrow \frac{\alpha \Delta t}{\Delta x^2} \leq \frac{1}{2}$$

7. The **harmonic series** $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots$ is known to diverge to $+\infty$. The nth partial sum approaches $+\infty$ at the same rate as $\ln(n)$. **Euler's constant** is defined to be

$$\gamma = \lim_{n\to\infty}\left[\sum_{k=1}^{n}\frac{1}{k} - \ln(n)\right] \approx 0.57721$$

If your computer ran a program for a week based on the pseudocode

```
**real** $s, x$
$x \leftarrow 1.0$;     $s \leftarrow 1.0$
**repeat**
    $x \leftarrow x + 1.0$;     $s \leftarrow s + 1.0/x$
**end repeat**
```

what is the largest value of s it would obtain? Write and test a program that uses a loop of 5000 steps to estimate Euler's constant. Print intermediate answers at every 100 steps.

8. (Continuation) Prove that Euler's constant, γ, can also be represented by

$$\gamma = \lim_{m\to\infty}\left[\sum_{k=1}^{m}\frac{1}{k} - \ln\left(m + \frac{1}{2}\right)\right]$$

Write and test a program that uses $m = 1, 2, 3, \ldots, 5000$ to compute γ by this formula. The convergence should be more rapid than that in the preceding computer problem. (See the article by De Temple [1993].)

[a]**1.** Write a routine for computing the two roots x_1 and x_2 of the quadratic equation $f(x) = ax^2 + bx + c = 0$ with real constants a, b, and c and for evaluating $f(x_1)$ and $f(x_2)$. Use formulas that reduce roundoff errors and write efficient code. Test your routine on the following (a, b, c) values: $(0, 0, 1)$; $(0, 1, 0)$; $(1, 0, 0)$; $(0, 0, 0)$; $(1, 1, 0)$; $(2, 10, 1)$; $(1, -4, 3.99999)$; $(1, -8.01, 16.004)$; $(2 \times 10^{17}, 10^{18}, 10^{17})$; and $(10^{-17}, -10^{17}, 10^{17})$.

2. (Continuation) Write and test a routine for solving a quadratic equation that may have complex roots.

3. Alter and test the pseudocode in the text for computing $x - \sin x$ by using nested multiplication to evaluate the series.

6. Write a procedure to compute $f(x) = \sin x - 1 + \cos x$. The routine should produce nearly full machine precision for all x in the interval $[0, \pi/4]$. *Hint:* The trigonometric identity $\sin^2\theta = \frac{1}{2}(1 - \cos 2\theta)$ may be useful.

7. Write a procedure to compute $f(x, y) = \int_1^x t^y \, dt$ for arbitrary x and y. *Note:* Notice the exceptional case $y = -1$ and the numerical problem *near* the exceptional case.

8. Suppose that we wish to evaluate the function $f(x) = (x - \sin x)/x^3$ for values of x close to zero.

- **a.** Write a routine for this function. Evaluate $f(x)$ sixteen times. Initially, let $x \leftarrow 1$, and then let $x \leftarrow \frac{1}{10}x$ fifteen times. Explain the results. *Note:* L'Hôpital's rule indicates that $f(x)$ should tend to $\frac{1}{6}$. Test this code.
- **b.** Write a function procedure that produces more accurate values of $f(x)$ for all values of x. Test this code.

9. Write a program to print a table of the function $f(x) = 5 - \sqrt{25 + x^2}$ for $x = 0$ to 1 with steps of 0.01. Be sure that your program yields full machine precision, but do not program the problem in double precision. Explain the results.

[a]**10.** Write a routine that computes e^x by summing n terms of the Taylor series until the $n + 1$st term t is such that $|t| < \varepsilon = 10^{-6}$. Use the reciprocal of e^x for negative values of x. Test on the following data: 0, +1, −1, 0.5, −0.123, −25.5, −1776, 3.14159. Compute the relative error, the absolute error, and n for each case, using the exponential function on your computer system for the exact value. Sum no more than 25 terms.

11. (Continuation) The computation of e^x can be reduced to computing e^u for $|u| < (\ln 2)/2$ only. This algorithm removes powers of 2 and computes e^u in a range where the series converges very rapidly. It is given by

$$e^x = 2^m e^u$$

where m and u are computed by the steps

$$\begin{array}{ll} z \leftarrow x/\ln 2; & m \leftarrow \text{integer}\,(z \pm \tfrac{1}{2}) \\ w \leftarrow z - m; & u \leftarrow w \ln 2 \end{array}$$

Here the minus sign is used if $x < 0$ because $z < 0$. Incorporate this range reduction technique into the code.

12. (Continuation) Write a routine that uses range reduction $e^x = 2^m e^u$ and computes e^u from the even part of the *Gaussian continued fraction*; that is,

$$e^u = \frac{s + u}{s - u} \quad \text{where} \quad s = 2 + u^2\left(\frac{2520 + 28u^2}{15120 + 420u^2 + u^4}\right)$$

Test on the data given in Computer Problem 2.2.10. *Note:* Some of the computer problems in this section contain rather complicated algorithms for computing various intrinsic functions that correspond to those actually used on a large mainframe computer system. Descriptions of these and other similar library functions are frequently found in the supporting documentation of your computer system.

[a]**15.** In the theory of Fourier series, some numbers known as **Lebesgue constants** play a role. A formula for them is

$$\rho_n = \frac{1}{2n+1} + \frac{2}{\pi}\sum_{k=1}^{n}\frac{1}{k}\tan\frac{\pi k}{2n+1}$$

Write and run a program to compute $\rho_1, \rho_2, \dots, \rho_{100}$ with eight decimal digits of accuracy. Then test the validity of the inequality

$$0 \leqq \frac{4}{\pi^2}\ln(2n+1) + 1 - \rho_n \leqq 0.0106$$

16. Compute in double or extended precision the following number:

$$x = \left[\frac{1}{\pi}\ln(6\,40320^3 + 744)\right]^2$$

What is the point of this problem? (See Good [1972].)

17. Write a routine to compute $\sin x$ for x in radians as follows. First, using properties of the sine function, reduce the range so that $-\pi/2 \leqq x \leqq \pi/2$. Then if $|x| < 10^{-8}$, set $\sin x \approx x$; if $|x| > \pi/6$, set $u = x/3$, compute $\sin u$ by the formula below, and then set $\sin x \approx [3 - 4\sin^2 u]\sin u$; if $|x| \leqq \pi/6$, set $u = x$ and compute $\sin u$ as follows:

$$\sin u \approx u\left[\frac{1 - \left(\dfrac{29593}{2\,07636}\right)u^2 + \left(\dfrac{34911}{76\,13320}\right)u^4 - \left(\dfrac{4\,79249}{1\,15113\,39840}\right)u^6}{1 + \left(\dfrac{1671}{69212}\right)u^2 + \left(\dfrac{97}{3\,51384}\right)u^4 + \left(\dfrac{2623}{16444\,77120}\right)u^6}\right]$$

Try to determine whether the sine function on your computer system uses this algorithm. *Note:* This is the Padé rational approximation for sine.

19. Write a routine to compute the tangent of x in radians, using the algorithm below. Test the resulting routine over a range of values of x. First, the argument x is reduced to $|x| \leqq \pi/2$ by adding or subtracting multiples of π. If we have $0 \leqq |x| \leqq 1.7 \times 10^{-9}$, set $\tan x \approx x$. If $|x| > \pi/4$, set $u = \pi/2 - x$; otherwise, set $u = x$. Now compute the approximation

$$\tan u \approx u\left(\frac{1\,35135 - 17336.106u^2 + 379.23564u^4 - 1.01186\,25u^6}{1\,35135 - 62381.106u^2 + 3154.9377u^4 + 28.17694u^6}\right)$$

Finally, if $|x| > \pi/4$, set $\tan x \approx 1/\tan u$; if $|x| \leqq \pi/4$, set $\tan x \approx \tan u$. *Note:* This algorithm is obtained from the *telescoped rational* and *Gaussian continued fraction* for the tangent function.

20. Write a routine to compute $\arcsin x$ based on the following algorithm, using telescoped polynomials for the arcsine. If $|x| < 10^{-8}$, set $\arcsin x \approx x$. Otherwise, if $0 \leqq x \leqq \frac{1}{2}$, set $u = x$, $a = 0$, and $b = 1$; if $\frac{1}{2} < x \leqq \frac{1}{2}\sqrt{3}$, set $u = 2x^2 - 1$, $a = \pi/4$, and $b = \frac{1}{2}$; if $\frac{1}{2}\sqrt{3} < x \leqq \frac{1}{2}\sqrt{2+\sqrt{3}}$, set $u = 8x^4 - 8x^2 + 1$, $a = 3\pi/8$, and $b = \frac{1}{4}$; if $\frac{1}{2}\sqrt{2+\sqrt{3}} < x \leqq 1$, set $u = \sqrt{\frac{1}{2}(1-x)}$, $a = \pi/2$, and $b = -2$. Now compute the approximation

$$\begin{aligned}\arcsin u \approx u\big(1.0 &+ \tfrac{1}{6}u^2 + 0.075u^4 + 0.04464\,286u^6 + 0.03038\,182u^8\\ &+ 0.02237\,5u^{10} + 0.01731\,276u^{12} + 0.01433\,124u^{14}\\ &+ 0.00934\,2806u^{16} + 0.01835\,667u^{18} - 0.01186\,224u^{20}\\ &+ 0.03162\,712u^{22}\big)\end{aligned}$$

Finally, set $\arcsin x \approx a + b\arcsin u$. Test this routine for various values of x.