Numerical Accuracy and Floating-Point Maths

David Grellscheid
based on slides by Axel Kohlmeyer, Temple U.

Before computations:

Modelling: neglecting certain properties
Empirical data: not every input is known perfectly

Previous computations: data may be taken from other (error-prone) numerical methods

Sloppy programming (e.g. inconsistent conversions)

During computations:

Truncation: a numerical method approximates a continuous solution

Rounding: computers offer only finite precision in representing real numbers

Computing the surface of the earth using

$$A = 4\pi r^2$$

This involves several approximations:

Modelling: the earth is not exactly a sphere

Measurement: earth's radius is an empirical number

Truncation: the value of π is truncated

Rounding: all numbers used are rounded due to arithmetic operations in the computer

Total error is the sum of all errors; usually one will dominate

Computing the surface of the earth using

$$A = 4\pi r^2$$

This involves several approximations:

Modelling: the earth is not exactly a sphere

Measurement: earth's radius is an empirical number

Truncation: the value of π is truncated

Rounding: all numbers used are rounded due to arithmetic operations in the computer

Total error is the sum of all errors; usually one will dominate

Real numbers have unlimited accuracy
On a computer, need to represent them in finite width

One option: fixed point numbers

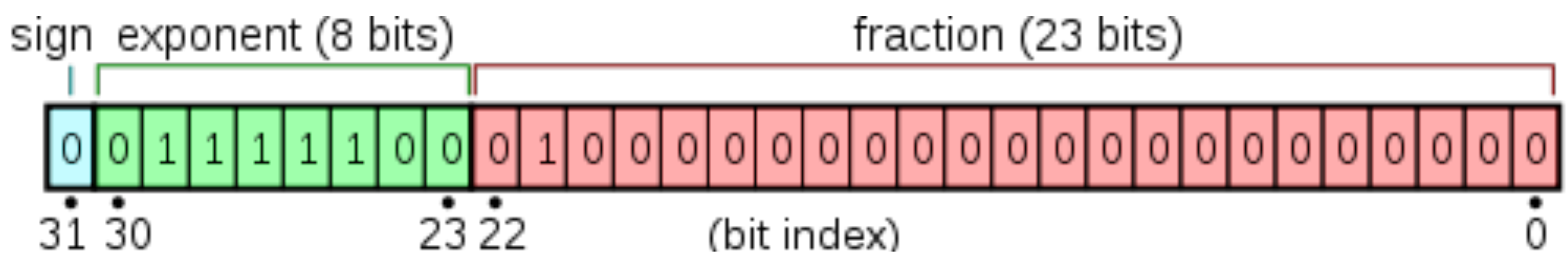
16-bit fixed: range ±32768, step size 1

12/4-bit fixed: range ±2048, step size 0.0625

Need wider range in the same number of bits, keeping reasonable precision.
Relative precision often sufficient

$$[\pm][1.\text{fraction}]\ 2^{[\text{exponent}]}$$

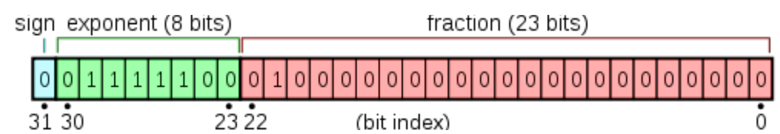
Single-precision floating point (float, 4 bytes)



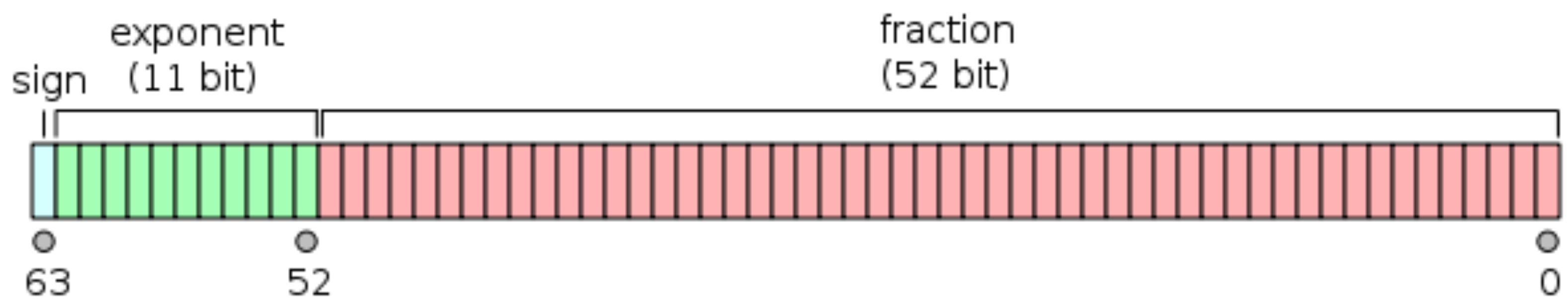
2^{23} = 8M available numbers between 1 and 2,
but only 4k numbers between 2048 and 2049

IEEE 754

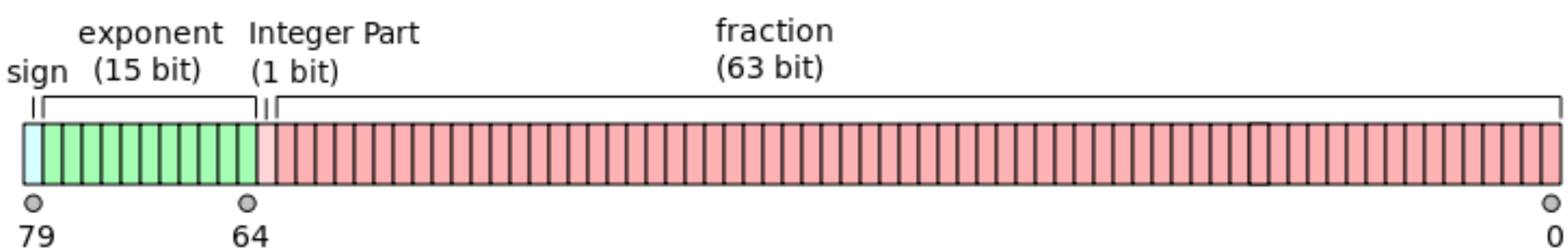
Single-precision floating point (float, 4 bytes)



Double-precision floating point (double, 8 bytes)



Extended-precision floating point (long double, 10 bytes)



Maths pitfalls

Gaps in representation

2^{23} = 8M available numbers between 1 and 2,
but only 4k numbers between 2048 and 2049

Almost all reals **cannot** be represented exactly

demo

Maths pitfalls

Gaps in representation

2^{23} = 8M available numbers between 1 and 2,
but only 4k numbers between 2048 and 2049

Almost all reals **cannot** be represented exactly

```
>>> 0.1+0.2
0.30000000000000004

>>> print '{:10.60f}'.format(0.1)
0.100000000000000005551115123125782702118158340454101562500000

>>> print '{:10.60f}'.format(0.2)
0.200000000000000011102230246251565404236316680908203125000000

>>> print '{:10.60f}'.format(0.3)
0.299999999999999988897769753748434595763683319091796875000000


>>> print '{:10.60f}'.format(0.1+0.2)
0.300000000000000044408920985006261616945266723632812500000000
```

Maths pitfalls

Gaps in representation

2^{23} = 8M available numbers between 1 and 2,
but only 4k numbers between 2048 and 2049

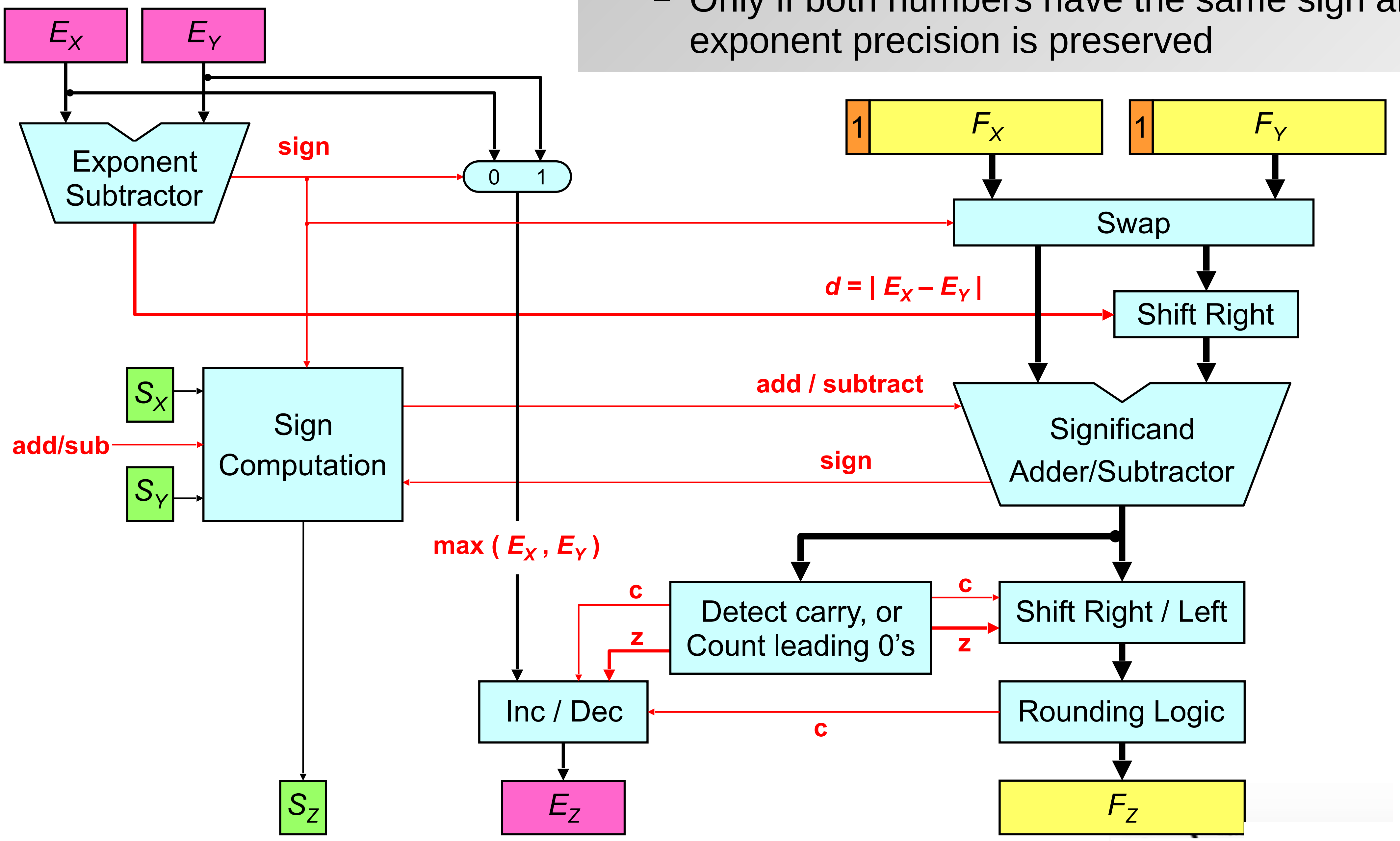
Almost all reals **cannot** be represented exactly

```
>>> 0.1+0.2
0.30000000000000004

>>> print '{:10.60f}'.format(0.1)
0.1000000000000000055511151231257827021

>>> print '{:10.60f}'.format(0.2)
0.200000000000000011102230246251565404236316680908203125000000

>>> print '{:10.60f}'.format(0.3)
0.299999999999999988897769753748434595763683319091796875000000


>>> print '{:10.60f}'.format(0.1+0.2)
0.300000000000000044408920985006261616945266723632812500000000
```

```
>>> 0.1 + 0.2 == 0.3
False
```

Addition:

- Right bitshift mantissa and increment exponent of smaller number until both exponents are the same
- Add mantissa of both numbers and bitshift until mantissa is between 1.0 and 2.0 again
- Only if both numbers have the same sign and the same exponent precision is preserved



Maths pitfalls

FP maths is commutative, **but not associative**

Value1	Value2	Value3	Value4	Sum
1.0E+30	-1.0E+30	9.5	-2.3	7.2
1.0E+30	9.5	-1.0E+30	-2.3	-2.3
1.0E+30	9.5	-2.3	-1.0E+30	0

the result of a summation depends on the order of how the numbers are summed up

results may change significantly, if a compiler changes the order of operations for optimisation

prefer adding numbers of same magnitude

avoid subtracting very similar numbers

Ill-conditioned matrices

$$1.0000\,x + 1.0000\,y = 2.0000$$
$$1.0000\,x + 1.0001\,y = 2.0000$$

$$1.0000\,x + 1.0000\,y = 2.0000$$
$$1.0000\,x + 1.0001\,y = 2.0001$$

$$\frac{x}{1000} + y = 1$$

$$x + y = 2$$

$$\frac{x}{1000} + y = 1$$

$$x + y = 2$$

$$x = \frac{1000}{999}$$

$$y = \frac{998}{999}$$

$$\frac{x}{1000} + y = 1$$

$$x + y = 2$$

$$x = \frac{1000}{999}$$

$$y = \frac{998}{999}$$

$$\frac{x}{1000} + y = 1$$

$$-999y = -998$$

$$\frac{x}{1000} + y = 1$$

$$x + y = 2$$

$$x = \frac{1000}{999}$$

$$y = \frac{998}{999}$$

$$\frac{x}{1000} + y = 1$$

$$-999y = -998$$

$$\frac{x}{1000} + y = 1$$

$$y = 1.00$$

$$\frac{x}{1000} + y = 1$$

$$x + y = 2$$

$$x = \frac{1000}{999}$$

$$y = \frac{998}{999}$$

$$\frac{x}{1000} + y = 1$$

$$-999y = -998$$

$$\frac{x}{1000} + y = 1$$

$$y = 1.00$$

$$\frac{999}{1000}y = \frac{998}{1000}$$

$$x + y = 2$$

$$\frac{x}{1000} + y = 1 \qquad x = \frac{1000}{999}$$

$$x + y = 2 \qquad y = \frac{998}{999}$$

$$\frac{x}{1000} + y = 1$$

$$-999y = -998$$

$$\frac{999}{1000}y = \frac{998}{1000}$$

$$x + y = 2$$

$$\frac{x}{1000} + y = 1$$

$$y = 1.00$$

$$1.00\,y = 1.00$$

$$x + y = 2$$

Japan J. Indust. Appl. Math., 26 (2009), 249–277

Inversion of Extremely Ill-Conditioned Matrices in Floating-Point

Siegfried M. Rump

$$A_4 = \begin{pmatrix} -5046135670319638 & -3871391041510136 & -5206336348183639 & -6745986988231149 \\ -640032173419322 & 8694411469684959 & -564323984386760 & -2807912511823001 \\ -16935782447203334 & -18752427538303772 & -8188807358110413 & -14820968618548534 \\ -1069537498856711 & -14079150289610606 & 7074216604373039 & 7257960283978710 \end{pmatrix}$$

Japan J. Indust. Appl. Math., 26 (2009), 249–277

Inversion of Extremely Ill-Conditioned Matrices in Floating-Point

Siegfried M. Rump

$$A_4 = \begin{pmatrix} -5046135670319638 & -3871391041510136 & -5206336348183639 & -6745986988231149 \\ -640032173419322 & 8694411469684959 & -564323984386760 & -2807912511823001 \\ -16935782447203334 & -18752427538303772 & -8188807358110413 & -14820968618548534 \\ -1069537498856711 & -14079150289610606 & 7074216604373039 & 7257960283978710 \end{pmatrix}$$

$$\mathtt{inv}_{\mathrm{fl}}(A_4) = \begin{pmatrix} -3.11 & -1.03 & 1.04 & -1.17 \\ 0.88 & 0.29 & -0.29 & 0.33 \\ -2.82 & -0.94 & 0.94 & -1.06 \\ 4.00 & 1.33 & -1.34 & 1.50 \end{pmatrix}$$

Japan J. Indust. Appl. Math., 26 (2009), 249–277

Inversion of Extremely Ill-Conditioned Matrices in Floating-Point

Siegfried M. Rump

$$A_4=\begin{pmatrix} -5046135670319638 & -3871391041510136 & -5206336348183639 & -6745986988231149 \\ -640032173419322 & 8694411469684959 & -564323984386760 & -2807912511823001 \\ -16935782447203334 & -18752427538303772 & -8188807358110413 & -14820968618548534 \\ -1069537498856711 & -14079150289610606 & 7074216604373039 & 7257960283978710 \end{pmatrix}$$

$$\mathtt{inv}_{\mathrm{fl}}(A_4)=\begin{pmatrix} -3.11 & -1.03 & 1.04 & -1.17 \\ 0.88 & 0.29 & -0.29 & 0.33 \\ -2.82 & -0.94 & 0.94 & -1.06 \\ 4.00 & 1.33 & -1.34 & 1.50 \end{pmatrix}$$

$$\mathrm{fl}(A_4^{-1})=\begin{pmatrix} 8.97\cdot10^{47} & 2.98\cdot10^{47} & -3.00\cdot10^{47} & 3.37\cdot10^{47} \\ -2.54\cdot10^{47} & -8.43\cdot10^{46} & 8.48\cdot10^{46} & -9.53\cdot10^{46} \\ 8.14\cdot10^{47} & 2.71\cdot10^{47} & -2.72\cdot10^{47} & 3.06\cdot10^{47} \\ -1.15\cdot10^{48} & -3.84\cdot10^{47} & 3.85\cdot10^{47} & -4.33\cdot10^{47} \end{pmatrix}$$