Computers making errors?

Surely you must be joking Mr Feynman!

Things to talk about

1. Look back at Random numbers
2. Computational Errors

Rounding, significant figures, word sizes, conversion, formulas, propagation

3. Common programming gotchas

Random numbers and Simulation

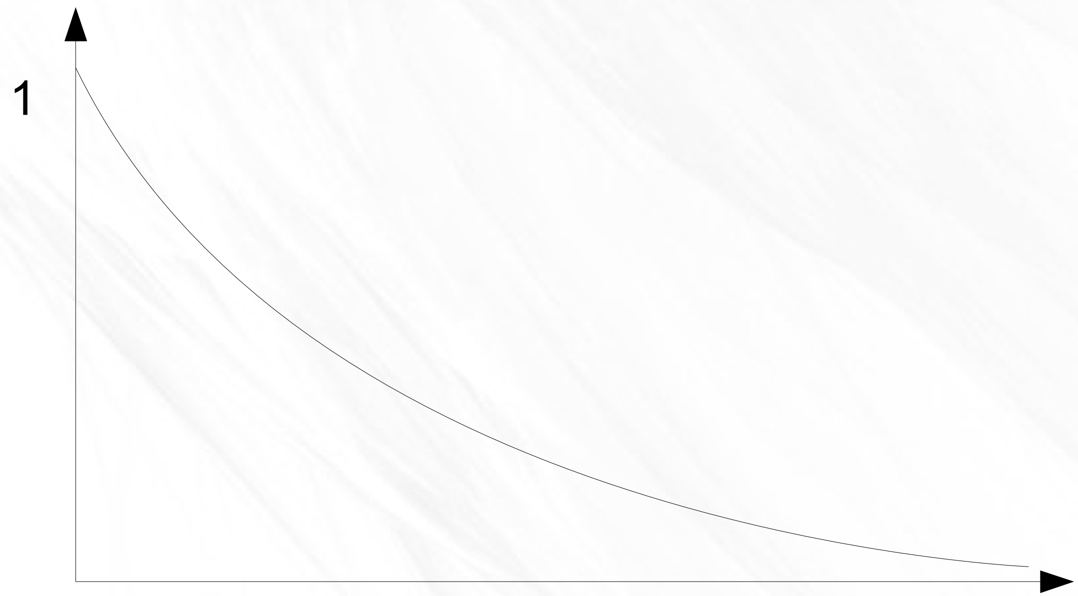
- Key points:
  - Computers can't make true random numbers
  - Only “pseudo-random” numbers
  - There are lots of ways of making random numbers, and different qualities
  - Random sequences of numbers should have the properties: uniformity, independence, summation, duplication.
  - There are ways of testing how random they are (NIST & Diehard tests)

Random numbers and Simulation

- Key points:
  - Ranges. [0,1] [2,3). Round-bracket means except the adjacent number, and square means including the adjacent number
  - Distributions – Continuous, discrete, but also uniform, normal, exponential, stable, and Levy distributions

Generating “Levy” Flights

- Take a continuous random number **r in [0,1)**
- And compute -ln(r) / 0.3



Programming your own RNG

```
unsigned int mw, mz;  // must be global
int main() {
  float f;
  // load mw and mz - these two numbers make up the seed
  mw = 35;
  mz = 478;
  f = (float) GetUniform();
  printf("%1.2f ", f);
}

unsigned int GetUint() {
  mz = 36969 * (mz & 65535) + (mz >> 16);
  mw = 18000 * (mw & 65535) + (mw >> 16);
  return (mz << 16) + mw;
}



double GetUniform() {
// returns a double in the open interval (0, 1)
  unsigned int u;
  u = GetUint();
  return (u + 1.0) * 2.328306435454494e-10;
}
```

Computational errors

- It's important to get it right, especially when simulating things like planes...
- Like if a simulation says that a plane with 1 wing will work just fine.

Computational errors

- Kinds of errors:
  - Rounding errors
  - Meaningless significant figures
  - Word sizes
  - Conversion errors
  - Human errors
  - Formula errors
  - Propagation errors
  - Other common programming errors

Rounding errors

- Many real numbers are infinitely long:

  4 / 3 = 1.3333333....

- This is called a repeating (or recurring decimal) because 3 is repeated forever.

  9 / 11 = 0.81818181.... 81 repeated forever
  pi = 3.1415926535..... Goes on forever

- In order to work with these numbers, you have to stop them at some point. This is known as **rounding**. Usually 0-4 rounds down, 5-9 up.

Rounding errors

1.33333333 could round to 1.667 (round up)

3.14125926535 could round to 3.14 (down)

0.81818181 could round to 0.82 or 0.818

- Another way is **truncation** used in C to convert float to int. Digits are just cut off:

1.333333... could be truncated to 1.33

0.818181... could be truncated to 0.81

Rounding errors

- To calculate that round-off error in %:

  Original value = 23.764462

  Rounded value = 23.764

  Round-off error = 0.000462

  % round-off error = 0.00194%

  (100 * (error / original))

Significant Figures

- The significant figures of a number are those digits that carry meaning.
- Excludes leading or trailing zeroes
- Doesn't matter where the decimal point is (s.f. = significant figures)

75684.3195 (9 s.f.)

23.764462 (8 s.f.)

0.0003786 (4 s.f.)

Significant Figures

- Alternative is to work to a fixed number of decimal places (d.p. = decimal places):

  75684.3195 changes to 75684.32 (2 d.p.)

  23.764462 changes to 23.7645 (4 d.p.)

  0.0003786 changes to 0.00 (2 d.p.)
- Is that last one useful?

Meaningless Significant Figures

- Two reasons for meaningless significant figures:
  - Assuming an approximation is accurate.

    A 100,000,005 year old skeleton.
  - Mathematical operations

    Theoretically, there are always infinite d.p.'s.
  - 37.26 * 0.02146 = 0.7995996 (7 s.f.)
  - Should round that to the original s.f. count:

    0.7995996 rounds to 0.7996 (4 s.f.)

Word Size errors

- Word size of a computer dictates how many s.f. are available for floating point operations
- Word size of 32 bits there are approx. 7
- Word size of 64 bits there are approx. 15

Conversion Errors

- Computers store numbers in binary
- Some numbers become infinitely long when stored in binary:

  0.2 converts to 0.0011001100110011

  (0011 repeats forever)

Human errors

- The initial data might be wrong
- Maybe wrong measurements.
- Garbage-in, garbage out!

Formula errors

- Some algorithms might be infinitely long
- To use them you have to stop them at some point (this introduces an error)
- Formula for sin(x) is:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

- Try it for x=pi/2 (radians) = 90 deg
- The more terms, the more accurate the answer, but it will never 100% correct.

Propagation of Errors

- Errors propagate rapidly and grow in size..

  Example:
- distance = 5.1m (error = 0.4m) (max is 5.5m)
  - 0.4 / 5.1 = 7.8% error
- Time = 0.4s (error = 0.1s) (min is 0.3s)
  - 0.1 / 0.4 = 25% error

Propagation of Errors

- Velocity = distance / time
- Using the measured figures:
  - 5.1m / 0.4s = 12.75m per second
- Using the figures with error we get
  - 5.5m / 0.3s = 18.333m per second
- Error = 18.333 – 12.75 = 5.583
- 5.583 / 12.75 = 43.8% error

Common programming errors

What does this print? The result should be 0.2 * 1000 which is 200.0

```
int main() {
  float val = 0.2;
  float tot = 0.0;
  int i;
  for (i = 0; i < 1000; i++) {
    tot = tot + val;
  }
  printf("tot is %f", tot);
} //tot is 199.998093
```

Common programming errors

What would happen if the next line in the program was this:

```
if (tot == 200.0) {
```

Common programming errors

What does this print? The result should be 5000.0001

```
int main() {

  float num1 = 5000.0;
  float num2 = 0.0001;
  float result = num1 + num2;

  printf("%f + %f = %f", num1, num2,
result);

}//5000.00000 + 0.00010 = 5000.00000
```

Common programming errors

- The last one was known as **subtractive cancellation**
- Adding a big number to a small number
- Big problems when adding a series of numbers
- Solve it by sorting from smallest to largest, and then add them in that order