ACCURACY AND SPEED

Ari Maller

- There are two main considerations for most computer algorithms which are often in tension with each other; accuracy and speed.
- Accuracy refers to the how close the computed number is to what one would get analytically when possible or compared to a more accurate calculation if possible.
- Speed refers to the time (or CPU time) it takes to perform the calculation.
- Usually accuracy and speed are in conflict in that you could perform a more accurate calculation but it would take longer and you can get a faster calculation but it is less accurate.

MACHINE PRECISION

- **Floats are stored on a computer using a fixed number of bits.**
- **Single precision:**
    - Sign: 1 bit; exponent: 8 bits; significand: 24 bits (23 stored) = 32 bits
    - Range: 2^7 -1 in exponent (because of sign) = 2^{127} multiplier ~ 10^{38}
    - Decimal precision: ~6 significant digits
- **Double precision:**
    - Sign: 1 bit; exponent: 11 bits; significand: 53 bits (52 stored) = 64 bits
    - Range: 2^{10}-1 in exponent = 2^{1023} multiplier ~ 10^{308}
    - Decimal precision: ~15 significant digits

EXERCISE

- Find the machine precision of your computer.

```
x=1.0
eps=1.0
while not x+eps==x:
    eps=0.5*eps
print(2*eps)
```

- This code will stop when adding eps to x doesn’t change the value of x.

OVERFLOW/UNDERFLOW ERRORS

- The computer uses a finite number of bytes to represent a number. This means there is a biggest possible floating point number the computer can represent.
- In Python this is about 10^{308}. If we tried to do y=10*1e308 in Python the value of y would be set to inf. This is called an overflow error.
- There is also a smallest possible float on the computer. If you try to make a float smaller than this value it will be set to 0.0 and you will get an underflow error.
- In most languages integers also have maximum values, which is why there are many types of integer variables. Python simply allocates more memory to store an integer, until you run out of memory.

ACCURACY - ROUNDING ERROR

- A finite number of bytes means that the value of floats can not be kept with infinite accuracy. The value of π has been determined do billions of digits of accuracy, but the computer will use a truncated value of π with as many digits as it uses to hold any float value.
- Since the computer stores a number in binary it won't even necessarily store the same value as you enter for a float.
- You should never check for a floats exact number x==0.1 because instead x might equal 0.10000000000000555. Instead check with an error

(abs(x-2.50) < ε)

- The rounding error behaves much like a measurement error in a physics lab. Importantly

$$\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$$

EXERCISE

- 0.1 is not represented exactly in binary

```
b=0.1

print(type(b))

print("{:30.20}".format(b))

import sys

sys.float_info
```

ACCURACY - ROUNDING ERROR

- It is usually a good assumption to consider the error to be a random number with standard deviation σ=Cx, where C is called the error constant and is 10^{-16} in Python.
- From this we can see if we perform N summations the error would be

$$\sigma^2 = \sum_{i=1}^{N} C^2 x_i^2 = C^2 N \bar{x}$$

- We see that the error increases as √N which means the fractional error **decreases** as √N, which in good. So what's the problem? Well in reality you may be adding numbers of very different sizes, which doesn't scale as well, but more importantly you may be performing subtraction.

EXERCISE

- difference of two numbers

  x=1

  y=1+$10^{-14}\sqrt{2}$

  print(1e14*(y-x))

  print(np.sqrt(2))

ACCURACY - ROUNDING ERROR

- Subtraction can lead to answers that are widely incorrect. If x=1000000000000000 and y=1000000000000001.25, x-y will give 1. instead of 1.25.
- Consider trying to evaluate exp(-24) using a Taylor series.

$$e^x \approx S(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!}$$

- If we compute S(-24) by adding terms until the term is less than machine precision – We find
- S(-24) = 3.44305354288101977E-007
- But exp(-24) = 3.77513454427909773E-011
- **This error is vastly bigger than the answer**

ACCURACY - ROUNDING ERROR

- What went wrong? The calculation involves adding alternatively larger positive and negative numbers, <u>subtraction of large numbers</u>. Rounding error is huge!
- If instead we could recognize that
- exp(-24)=exp(-1)**24 -> S(-24) = S(-1)**24
- S(-1) is well behaved, since each term is smaller in absolute magnitude than the previous.
- S(-1) = 0.36787944117144245
- S(-1)24 = 3.77513454427912681E-011
- exp(-24) = 3.77513454427909773E-011

VOLUME OF A SHELL

- Another common example, often we might need the volume of a shell in a calculation.

$$V = \frac{4}{3}\pi(r_1^3 - r_2^3)$$

- This relies on the cancelation of two big numbers. Danger!!
- Instead rewriting the formula as

$$V = \frac{4}{3}\pi\Delta r(r_1^2 + r_1 r_2 + r_2^2)$$

- is much safer. Try it for some large values of r.

EXERCISE 4.2

- Write a program that takes as input 3 numbers; a, b and c and prints out two solutions to the quadratic equation.
- Use your program to compute the solution to $0.001x^2 + 1000x + 0.001 = 0$
- There is another way to write the quadratic equation if you multiply the numerator and denominator by $-b \mp \sqrt{b^2-4ac}$, you'll get

$$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

- Add further lines to your program to also use this alternative solution. Do you get the same value for x? why?

SPEED

- Besides not having infinite memory a computer also does not have infinite speed. While most operations happen so fast they seem to take zero time, they in fact take a finite about of time.
- This is readily discernible if you do millions or billions of calculations, instead of one.
- A million calculations can be done in around 1 second.
- However, this means a billion calculations will take around 20 minutes.
- A trillion calculations would take 300 hours or 12 days.
- While the computer is incredibly fast, it is not infinitely fast. There is an upper limit to the number of calculations one can do in a reasonable time (where reasonable usually has to do with when you need the calculation by).

SPEED

- In many cases increasing the number of calculations does not increase the accuracy. Thus one wants to be careful. Often an increase in speed does not mean much of a loss of accuracy.
- One place where the number of operations increases very rapidly in matrix multiplication (or general manipulation).

```
C=np.zeros([N,N],float)
for i in range(N):
    for j in range(N):
        for k in range(N):
            C[i,j]+= A[i,k]*B[k,j]
```

- This code snippet does two operations, multiplication and addition calculation. This is done N times over k, N times over j and N times over i so a total of $2N^3$ operations.
- For N=1000 this is two billion operations. For N=2000 this is 16 billion operations. In practice it is complicated to handle matrices much larger than 1000×1000.

EXERCISE

- Quantum Harmonic Oscillator
- Let's write a program to solve the average energy in a quantum harmonic oscillator which has energy levels $E_n = hf(n+1/2)$.
- The average energy is given by

$$< E >= \frac{1}{Z}\sum_{n=0}^{\infty} E_n e^{-\beta E_n}$$

$$Z = \sum_{n=0}^{\infty} e^{-\beta E_n}$$

- taking hf=1, β=0.01 let us evaluate this formula using a thousand terms, a million terms and a billion terms.

TERMINOLOGY

- Overflow/Underflow Error - When a float becomes to big or small to be represented with the number of bytes allocated for floats. This is greater than 10^{308} and less than 10^{-308} in python.
- Rounding/Roundoff Error - An error caused because a float only contains a certain number of bytes and thus can't hold accuracy past a limit. This is 2×10^{-15} in python.
- Machine Precision - The value of the accuracy of floats. Again, 2×10^{-15} in python.