Lecture 5

Example of graphics w/ wave interference.

Dropping a pebble in water leads to waves radiating outward from where it was dropped.

Given drop location at x_1, y_1 distance from here is given by

$$r_1 = \sqrt{(x-x_1)^2 + (y-y_1)^2}$$

& sine wave for height is

$$\zeta_1(x,y) = \zeta_0 \sin(kr_1)$$

where ζ_0 is amplitude &

$k = 2\pi/\lambda$ is wavevector w/ λ wavelength

Dropping a pebble @ a different location would lead to

$$\xi_2(x,y) = \xi_0 \sin(kr_2)$$

w/ $r_2 = \sqrt{(x-x_2)^2 + (y-y_2)^2}$

superposition of waves is created leading to

$$\xi_3(x,y) = \xi_0 \sin(kr_1) + \xi_3 \sin(kr_2)$$

Given $\lambda = 5\,cm$, amplitude $= 1\,cm$ & centers of circles 20 cm apart, we can plot the waves

Bring up 1-ripples.py

The Mandelbrot set is a fractal, containing structure within structure.

Let c be a complex number

then for some $z \in \mathbb{C}$, compute

$$z' = z^2 + c$$

keep repeating calculation, plugging in z' as z.

Mandelbrot set M defined as

Let $c \in \mathbb{C}$ & $z = 0$ & iterate as above repeatedly. If $|z| > 2$ then $c \notin M$.

otherwise $c \in M$.

would need to iterate this infinitely many times, but in practice can just run a finite # of iterations.

Bring up 1-1-mandelbrot.py

Chap. 4 – Accuracy + speed

Computers have limitations,
both for storage & processing time
talk about these now because
they can seriously affect physics
calculations.

Variables & ranges

python cannot store numbers that
are arbitrarily large.
largest value for float is $\approx 10^{308}$
& $\approx -10^{308}$

real & imaginary parts of complexes
are limited by this as well.

can specify large numbers w/ "e" for exponent. 2e9 means 2×10^9

If value of variable exceeds largest value, we say it has "overflowed"

e.g. if x holds something close to 10^{308} & we write $y = 10 * x$

then y will overflow.

Python sets it to special value inf, & treats it as if it is infinity.

smallest # is 10^{-308}

If you go smaller, the # overflows & the computer sets it equal to zero.

For integers, Python uses arbitrary size allowable by your computer.

Comparison for factorial

Bring up 2-factorial-functionapy

Numerical error

Numbers like π or $\sqrt{2}$ can only be represented approximately, level of precision is 16 significant digits

For example,

π = 3.1415 9265 3589 7932 38...

in Python = 3.1415 9265 3589 793

---

difference = 0.0... 0.238...

difference is called rounding error

Numbers in a floating-point calculation are only guaranteed to have accuracy to 16 significant figures

execute print(1.1 + 2.2) in Python...

consequence ⟶

never use an if statement to test equality of floating point numbers.

E.g., never do

```
if x == 3.3:
    print(x)
```

should instead do

```
epsilon = 1e-12
if abs(x - 3.3) < epsilon:
    print(x)
```

choose ε appropriately for your situation

If we execute

```
from math import sqrt
x = sqrt(2)
```

we don't get $x = \sqrt{2}$ but rather $x + \varepsilon = \sqrt{2}$ or $x - \varepsilon = \sqrt{2}$

we cannot necessarily count on ε being small. How to model this error?

if a number x is accurate to 16 digits, then rounding error will have a typical size of $x/10^{16}$.

Because we're dividing up space into uniform intervals, simplest model is that error is a uniformly distributed random variable.

Recall that mean of uniform R.V. on $[a,b]$ is $\frac{1}{2}(a+b)$ & variance is $\sigma^2 = \frac{1}{12}(b-a)^2$ so that st. dev. is $\sigma = \frac{1}{\sqrt{12}}(b-a)$

$\Rightarrow$ st. dev. for error for a number x w/ 16 significant digits will be

$\sigma = \epsilon x$ where $\epsilon \approx 10^{-16}$

error can grow if we add or subtract, multiply or divide numbers.

If we add two numbers, then errors add & by assuming independent errors, std. dev.'s sum

$\sigma^2 = \sigma_1^2 + \sigma_2^2$ so then by the above

$\sigma = \sqrt{\epsilon^2 x_1^2 + \epsilon^2 x_2^2} = \epsilon \sqrt{x_1^2 + x_2^2}$

results extend for addition of

N numbers $x_1, \ldots, x_N$:

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

$$= C^2 N \overline{x^2} \quad \text{where}$$

$$\overline{x^2} = \frac{1}{N} \sum_{i=1}^{N} x_i^2$$

is mean-square value of sequence

So std. dev. is

$$\sigma = C\sqrt{N}\ \overline{x^2}$$

$\Rightarrow$ as N increases so does error, but goes as $\sqrt{N}$.

As an example, suppose that you are adding

$x = 10\ldots\ldots00$

$y = 10\ldots01.2345\ 6789\ldots$

Then computer will represent as

$$x = 10\cdots\infty$$

$$y = 10\cdots01.2$$

$$\Rightarrow \quad y - x = 1.2$$

very different from true value
ϵ can lead to problems

Another example:

take $x = 1$

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

$$\Rightarrow \quad 10^{14}(y - x) = \sqrt{2}$$

Bring up 3-error.py

calculation only accurate to first decimal place.

Example w/ calculating derivatives

Mathematical definition of derivative of $f(x)$:

$$\frac{df(x)}{dx} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

can get a reasonable approximation by making δ small (but not too small)

E.g., suppose $f(x) = x(x-1)$

+ we want value of derivative @ $x = 1$.

Then $\left.\frac{df(x)}{dx}\right|_{x=1} = \left.2x - 1\right|_{x=1} = 1$

can write computer program for this

Bring up 3-1 derivative.py

Program speed

Computers are limited in speed. However, they are fast & can handle a million operations easily. A billion could take minutes or hours, so author suggests as a rough guide that a computer can handle a billion or less.

As example, consider quantum simple harmonic oscillator w/ energy levels

$E_n = \hbar\omega\left(n+\frac{1}{2}\right)$ for $n = 0, 1, 2, \ldots$

Average energy @ temperature T

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

where $\beta = \frac{1}{k_B T}$ is inverse temperature

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

partition function

Suppose we want to calculate E when $k_B T = 100$.

Since we have an exponential falloff w/ increasing n, we can get a reasonable approximation by taking n large, say $n = 1000$

Take units $\hbar = \omega = 1$,

Bring up 4-qsho.py

Features of program:

1. constants are defined at the beginning
2. only one "for" to calculate two sums
3. Even though $e^{-\beta E_n}$ occurs in both sums,

can increase number of terms to get more accurate answer, but then there is a trade-off between time & accuracy.

Should do an estimate for how long it will take a calculation to run before doing it.

Another example: matrix multiplication

How to multiply two matrices A + B together?

Bring up 5-matrix-multiply.py

What is the complexity of this simple algorithm for matrix multiplication?

3 nested "for" loops

innermost one has N additions + multiplications, for a total of 2N operations

nested inside of two for loops that each go around N times, giving a total of $2 \cdot N^3$ operations

so if $N = 1000$, that would be two billion operations, which would require a few minutes of running time...

then if $N = 2000$, we need 16 billion ops & becomes more difficult.

There is an algorithm called Strassen's algorithm for matrix multiplication which brings complexity down to $N^{2.8}$, which can be much faster for large matrices. Area of open research to figure out what the optimal exponent is.

Best exponent known is 2.3729... but only useful for matrices that are too large to handle on present day computers

(this is due to Francois le Gall, quantum computational theorist)