Lecture 16

Fast Fourier transform

Recall that the DFT is

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right)$$

Python program that we gave before has a for-loop for each coefficient & N terms in the sum, implying N^2 ops to get all coefficients.

If we are not willing to wait for more than 1 billion ops, then we can do a DFT for

$$N^2 = 10^9 \Rightarrow N \approx 32000 \text{ samples}$$

This is not very much, just about one second of audio. The FFT is a much better way for doing this, discovered by Gauss

easiest to describe when # of samples is a power of two.

Let $N = 2^m$ w/ m an integer

Consider the sum in the DFT equation & divide it into 2 groups, even & odd terms

Consider even terms first, where $n = 2r$ w/ $r \in \{0, \ldots, \frac{1}{2}N-1\}$

then $$E_k = \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp\left(-i \frac{2\pi k(2r)}{N}\right)$$

$$= \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp\left(-i \frac{2\pi kr}{N/2}\right)$$

call this → $E_{k \bmod N/2}$ This is actually just another DFT w/ $N/2$ samples instead of N

look @ odd terms now:

$$\sum_{r=0}^{1/2N-1} y_{2r+1} \exp\left(-i \frac{2\pi k(2r+1)}{N}\right) = \rightarrow$$

$$= e^{-i2\pi k/N} \sum_{r=0}^{1/2N-1} y_{2r+1} \exp\left(-i\frac{2\pi kr}{N/2}\right)$$

$$= e^{-i2\pi k/N} O_k$$

where O_k is another DFT w/ $N/2$ samples
call this $O_{k \bmod N/2}$

So this means that

$$c_k = E_{k \bmod N/2} e^{-i2\pi k/N} O_{(k) \bmod (N/2)}$$

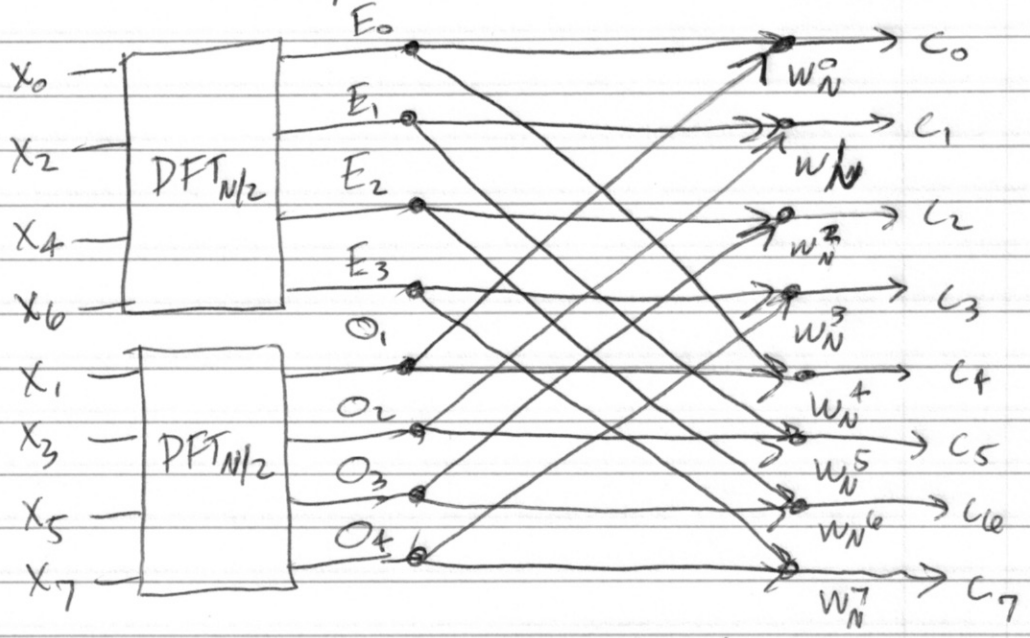
$$= E_{k \bmod N/2} W_N^k O_{k \bmod N/2}$$

We can write what we have written in terms of a diagram.

Let 

denote an N-point DFT. The outputs are the DFT of the input

Then to compute c_0, we do



famous "butterfly"

Thus the ~~same~~ diagram for the FFT.

It specifies the recursion &

the starting point is DFT_1 which is just the identity transformation.

At the ~~kth~~ ~~step~~ m^{th} stage there are $O(N)$ calculations.

There are $\log N$ stages, meaning that the ~~brute force~~ FFT requires $O(N \log N)$ operations Near linear!

Eg. if we have $N = 1$ million samples to process, then naive way requires

$N^2 = 10^{12}$ operations, which is nontrivial for a typical computer. if not practical

But $N \log N = 10^7$ which is reasonable (under one second)

Inverse DFT can be done in the same way.

FFT in Python provided by module numpy.fft

bring up 1-fft.py

rfft computes fft for a set of real numbers, only returns the 1st half because the other half are complex conjugates by symmetry

can also calculate Fourier transforms of complex data w/ fft & ifft

& 2D transforms w/ rfft2, irfft2, fft2, ifft2

also functions for higher dimensions

Chapter 8 – Solving ordinary differential equations

1st-order differential equations

Consider $\frac{dx}{dt} = \frac{2x}{t} + \frac{3x^2}{t^3}$

It is not separable & is also nonlinear, so we need to resort to the computer to solve it.

general form for a 1st-order differential equation is $\frac{dx}{dt} = f(x,t)$

to calculate a full solution, we need a boundary condition, e.g., the value of x @ one particular value of t (usually $t=0$).

Euler's method

Suppose we have to solve $\frac{dx}{dt} = f(x,t)$

& an initial condition.

Then Taylor expand $x(t+h)$ about t

$$x(t+h) = x(t) + h\frac{dx}{dt} + \frac{h^2}{2}\frac{d^2x}{dt^2} + \dots$$
$$= x(t) + h\,f(x,t) + O(h^2)$$

If we neglect $O(h^2)$ terms

then we get

$$x(t+h) = x(t) + h\,f(x,t)$$

So if we know x @ time t, we can just use this equation to iterate

if h is small enough, then this does pretty well.

called Euler's method

Example: $\frac{dx}{dt} = -x^3 + \sin t$

Bring up 2-euler.py

No one really uses Euler's method because there is a more accurate approach that requires very little extra effort to program. These are the Runge-Kutta methods.

The error in one step of the Euler method is to leading order

$$\frac{1}{2}h^2 \frac{d^2x}{dt^2}$$

Error for many steps ~~when~~ from $t=a$ to $t=b$ using step size h is then

$N = \frac{b-a}{h}$ values of t for steps one @ $t_k = a + kh$

+ x_k is corresponding value for x

Then the total error is

$$\sum_{k=0}^{N-1} \frac{1}{2} h^2 \left(\frac{d^2x}{dt^2}\right)_{\substack{x=x_k,\\ t=t_k}} = \frac{1}{2} h \left[\sum_{k=0}^{N-1} h \left(\frac{df}{dt}\right)_{\substack{x=x_k,\\ t=t_k}}\right]$$

$$\approx \frac{1}{2} h \int_a^b \frac{df}{dt}\, dt$$

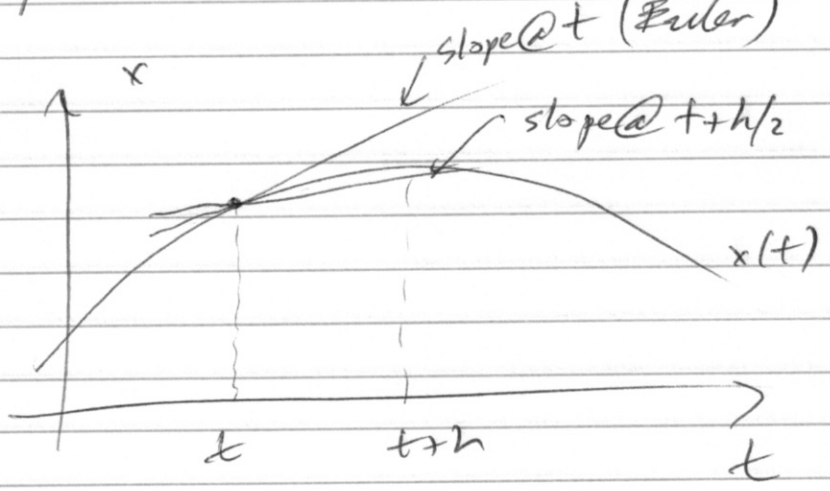
$$= \frac{1}{2} h \left[f(x(b), b) - f(x(a), a) \right]$$

approx. is good if h is small.

error is linear in h, so that if we make it smaller, then error is smaller.

Runge-Kutta method can be much better. (these are really a set of methods

Can express Euler's method in graphical form as



curve is $x(t)$, diff. eq. $\frac{dx}{dt} = f(x,t)$ says that slope of solution is equal to $f(x,t)$.

So Euler method extrapolates slope to time $t+h$, giving an estimate of $x(t+h)$

2nd order Runge-Kutta method is to use the slope @ midpoint $t+h/2$ to extrapolate to $t+h$

Perform a Taylor expansion of $x(t+h)$ about $t+\frac{1}{2}h$:

$$x(t+h) = x(t+1/2h) + \frac{1}{2}h\left(\frac{dx}{dt}\right)_{t+\frac{1}{2}h} + \frac{1}{8}h^2\left(\frac{d^2x}{dt^2}\right)_{t+\frac{1}{2}h} + O(h^3)$$

$x(t)$ has expansion

$$x(t) = x(t+\tfrac{1}{2}h) - \frac{1}{2}h\left(\frac{dx}{dt}\right)_{t+\frac{1}{2}h} + \frac{1}{8}h^2\left(\frac{d^2x}{dt^2}\right)_{t+\frac{1}{2}h} + O(h^3)$$

Subtract 2nd from 1st to get

$$x(t+h) = x(t) + h\left(\frac{dx}{dt}\right)_{t+\frac{1}{2}h} + O(h^3)$$

$$= x(t) + h\, f(x(t+\tfrac{1}{2}h), t+\tfrac{1}{2}h) + O(h^3)$$

$\Rightarrow$ that the order h^2 term has disappeared! & error term is now order h^3

Problem is that this approach requires an estimate of $x(t+\frac{1}{2}h)$ which we don't have (we only know $x(t)$). Instead approximate $x(t+\frac{1}{2}h)$ using Euler's method:

$$x(t+\tfrac{1}{2}h) = x(t) + \tfrac{1}{2}h f(x,t)$$

& then substitute. So we get

$$k_1 = h f(x,t)$$

$$k_2 = h f(x + \tfrac{1}{2}k_1, t + \tfrac{1}{2}h)$$

$$x(t+h) = x(t) + k_2$$

Final Runge-Kutta estimate

This is a "second order" method accurate to order h^2 w/ error of order h^3.

How do we know this when we approximated $x(t+\frac{1}{2}h)$ w/ Euler's method?

Consider that (T.E. $f(x+\frac{1}{2}k_1, t+\frac{1}{2}h)$ around $x(t+\frac{1}{2}h)$)

$$f(x(t)+\tfrac{1}{2}k_1, t+\tfrac{1}{2}h)$$

$$= f(x(t+\tfrac{1}{2}h), t+\tfrac{1}{2}h)$$

$$+\left[x(t)+\tfrac{1}{2}k_1 - x(t+\tfrac{1}{2}h)\right]\left(\frac{\partial f}{\partial x}\right)_{\substack{x(t+h/2),\\ t+h/2}}$$

$$+ O\left(\left[x(t)+\tfrac{1}{2}k_1 - x(t+\tfrac{1}{2}h)\right]^2\right)$$

From before we have

$$x(t+\tfrac{1}{2}h) = x(t) + \tfrac{1}{2}hf(x,t) + o(h^2)$$

$$= x(t) + \tfrac{1}{2}k_1 + o(h^2)$$

$$\Rightarrow x(t) + \tfrac{1}{2}k_1 - x(t+\tfrac{1}{2}h) = o(h^2) +$$

$$f(x(t)+\tfrac{1}{2}k_1, t+\tfrac{1}{2}h)$$

$$= f(x(t+\tfrac{1}{2}h), t+\tfrac{1}{2}h) + o(h^2)$$

$$\Rightarrow k_2 = hf(\dot{x}(t+\tfrac{1}{2}h), t+\tfrac{1}{2}h) + o(h^3)$$

so error introduced by Euler approximation is order h^3

Bring up 3-rk2.py

run for different #s of points = N = 10, 20, 50, 100

Can continue the Runge-Kutta method further. Take more Taylor expansions to cancel out higher order terms. Popular method is fourth order Runge-Kutta method which is still relatively simple to program:

$$k_1 = h f(x,t)$$

$$k_2 = h f(x + \tfrac{1}{2}k_1, t + \tfrac{1}{2}h)$$

$$k_3 = h f(x + \tfrac{1}{2}k_2, t + \tfrac{1}{2}h)$$

$$k_4 = h f(x + k_3, t + h)$$

$$x(t+h) = x(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

- accurate to terms of order h^4 & has error of order h^5

- derivation of this is complicated but the final equations are simple

- just 5 equations but 3 orders of h more accurate than Euler's method

- 4th order R-K is the method of choice for solving ODEs on computer.

Bring up 4_rk4.py

run w/ N = 10, 20, 50, 100

Finding solutions over an infinite range

sometimes want to go out to $t=\infty$

idea is to use a change of variables as before.

Let $u = \frac{t}{1+t}$ or $t = \frac{u}{1-u}$

so that $u \to 1$ as $t \to \infty$

re write diff. eq. w/ chain rule as

$$\frac{dx}{du}\frac{du}{dt} = f(x,t) \quad \text{or}$$

$$\frac{dx}{du} = \frac{dt}{du} f\left(x, \frac{u}{1-u}\right)$$

But $\frac{dt}{du} = \frac{1}{(1-u)^2} f\left(x, \frac{u}{1-u}\right)$

so $\frac{dx}{du} = \frac{1}{(1-u)^2} f\left(x, \frac{u}{1-u}\right)$

Let $g(x,u) = (1-u)^2 f\left(x, \frac{u}{1-u}\right)$

so $\frac{dx}{du} = g(x,u)$ & then just use techniques from before

Example: Suppose we want to solve

$$\frac{dx}{dt} = \frac{1}{x^2+t^2}$$ from $t=0$ to $t=\infty$ w/ $x=1$ @ $t=0$

then instead solve

$$g(x,u) = \frac{1}{(1-u)^2} \frac{1}{x^2 + \frac{u^2}{(1-u)^2}}$$

$$= \frac{1}{x^2(1-u)^2 + u^2}$$

So solve

$$\frac{dx}{du} = \frac{1}{x^2(1-u)^2+u^2}$$ from $u=0$ to $u=1$

Bring up 5-ode.nfopy