1 | base knowlege

1.1 | primitive root of unity

1.1.1 | a number r is a primitive nth root of unity iff n is the smallest counting number for which $r^n = 1$.

1.1.2 | `https://mathworld.wolfram.com/PrimitiveRootofUnity.html` source

1.2 | convolution theorem

1.2.1 | 'depends fundamentally on the convolution theorem, which provides an efficient way to compute the cyclic convolution of two sequences. It states that the cyclic convolution of two vectors can be found by taking the discreate fourier transform of each of them, multiplying the resulting vectors element by element, and then taking the inverse discrete fourier transform.'

2 | sources

2.1 | explanation of multiplication algorithm

2.2 | paper explaining the multiplication ANDO EMERENCIA (S1283936)

2.3 | FFT Medium Blog Post

2.4 | wikipedia on schonhage-strassen (multiplication algo)

3 | uses of FFT

3.1 | convert mixed signals into constituent sinusoids

3.2 | multiply polynomials using convolution theorem

3.3 | reduce matrix dimensionality

3.4 | audio processing (eg. bass boost, or radio denoising for eg. wifi)

3.5 | MRI machines? scan certain parts using different overlapping sinusoidal magnitudes of magnetic field

3.6 | microscope or astronomy image decomposition

3.7 | connection to Heisenberg uncertainty principle (`https://www.youtube.com/watch?v=MBnnXbOM5S4`)

3.7.1 | the fourier transform graph says something about correlation

1. eg. for a pure signal (sine 5Hz), a winding frequency of $\xi = 5.01$ will come out pretty high on the almost-fourier-transform graph, aka it is closely correlated

2. since the actual fourier transform isn't divided by signal length, longer signals will lead to higher (and steeper) peaks
3. these steeper peaks represent more certainty, and a shorter signal means fewer cycles means the signal has less time to balance itself out means the almost-fourier-transform will have shallower and wider peak means less certainty

3.7.2 | can also use the norm of the fourier transform (to capture both x and y bc complex)

3.7.3 | when using Doppler radar (in which a single pulse detects position normally and velocity using Doppler shift), this trade off the uncertainty principle shows up

1. since longer radar pulse introduce distance uncertainty and shorter radar pulses have the frequency uncertainty implied by the fourier transform as correlation above

3.7.4 | particle as a wave -> relativistic doppler effect ends up having something similar

4 | 3b1b video `https://www.youtube.com/watch?v=spUNpyF58BY`

4.1 | unmixing waves

4.1.1 | the added up ones seem needlessly complex for such a little amount of info

4.2 | rotating the wave around a circle

4.2.1 | aka: wave around the circle is polar coords: length = magnitude of wave at that point, offset = phase + some angular velocity (the 'rotation' frequency)

4.2.2 | there are two frequencies: 1. the frequency at which the vector goes around the circle 'winding frequency', and 2. the original and 'true' frequency of the wave

4.2.3 | when the frequencies match, all the high points are on the right and low points are on the left… question is how can we quantify this specialness

4.2.4 | center of mass as a function of the winding frequency

1. frequency of zero is high, and then it wobbles for a while until a frequency matches

4.3 | central construct

4.3.1 | original plot (intensity | time)

4.3.2 | winding chart (wound signal | signal, winding frequency)

4.3.3 | center-of-mass plot (x coord | winding frequency)

1. the spike at zero only happens because the original freq doesn't oscillate about zero

4.4 | he calls this the 'almost Fourier transform'

4.4.1 | additive: you can take the almost fourier transform first or you can take the sum first and you will get the same center-of-mass plot out

1. pause and ponder: multiple arrows going around the circle, tip to tail

4.5 | formalizing the 'center of mass'

4.5.1 | complex numbers: works well for 2d plane and rotation can be described by

$$e^{2\pi it}$$

by multiplying that t by a scalar, you can change the frequency:

$$e^{2\pi ift}$$

4.5.2 | actual formalization

1. convention: rotate in clockwise direction

$$e^{-2\pi ift}$$

2. let the original function be called $g(t)$, then scale by that for the 'vector following the original graph magnitude'

$$g(t)e^{-2\pi ift}$$

3. tracking 'center of mass': sample points and average them
   if N is the number of points that you sample and t_k is the k-th sampled point,

$$\frac{1}{N}\sum_{i=1}^{N} g(t_k)e^{-2\pi ift_k}$$

4. and if we want a more accurate sample, just take the limit to infinity

$$\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N} g(t_k)e^{-2\pi ift_k}$$

5. which is really the same as taking the integral

$$\frac{1}{t_2 - t_1} \int_{t_1}^{t_2} g(t) e^{-2\pi i f t} dt$$

6. but we don't actually need to divide by the time interval

$$\int_{t_1}^{t_2} g(t) e^{-2\pi i f t} dt$$

This means that when a frequency persists for a long time, it gets scaled more

5 | Reducible on FFT

5.1 | intro: its important and beautiful

5.2 | start with multiplying polynomials

5.2.1 | represent by list of coefficients in ascending order (index = degree of term) (polynomial representation)

5.2.2 | another option: two-point representation

1. extension: any degree n polynomial can be represented by $n + 1$ points uniquely
2. proof: write as a system of equations, matrixify, and we know that the matrix will be invertible
3. its a bijection, lets call it the value representation

5.2.3 | now to multiply, we can just take enough points on each polynomial and multiply those points to get $d_1 + d_2 + 1$ points on the product polynomial, and then solve for the actual equation

5.2.4 | only O(d) operations

5.2.5 | now, we need a function take coefficients to values and values to coefficients, this box is the fast fourier transform

5.3 | forward direction: evaluation

5.3.1 | naive: evaluating each point takes O(d) operations which means the total eval will be d^2, which is no better

5.3.2 | simpler problem

1. suppose $f(x) = x^2$, then picking points that are symmetric (bc even function) means you only have to evaluate half of them
2. can also do something similar for odd functions. so when you split a general polynomial into even terms and odd terms, then we can just eval each and get double the results

5.3.3 | he calls them $P_e(x^2)$ and $P_o(x^2)$, and they are polynomials of x^2 so deg 2?? toexpand

5.3.4 | so now we recurse

1. now it will be $O(n \log n)$

5.3.5 | a problem: we need to choose positive/negative pairs but since the recursed ones are squared, then everything will be positive

1. how to solve this problem… work over the complex numbers!
2. now what initial points do we want to choose… an example shows that it should be $x^n = 1$ for a fourth degree polynomial
3. so we want roots of unity, for some $n \leq d + 1$ and $n = 2^k$
4. how to write it:

   $$\omega = e^{\frac{2\pi i}{n}}$$

   Then each root of unity can be expressed as a power of ω

   evaluate P(x) at

   $$[1, \omega, \omega^2, \ldots, \omega^{n-1}]$$

5. why these
   (a) positive-negative paired: the point across the circle is the pair
   (b) when squared, the n roots of unity become the n/2 roots of unity, which still have points across the circle

5.3.6 | recursion time

1. base case: n = 1 -> P(1)
2. recurse
   (a) split into even/odd degree terms
   (b) recurse to get y_e, y_o
   (c) some math ($x_j = \omega^j$, $-\omega^j = \omega^{j+\frac{n}{2}}$, $y_e[j] = P_e(\omega^{2j}), y_o[j] = P_o(\omega^{2j})$) shows $P(\omega^j) = y_e[j] + \omega^j y_o[j], P(\omega^{j+\frac{n}{2}}) = y_e[j] - \omega^j y_o[j]$

5.3.7 | its clean to use d is a power of two, but there are impls that can handle others also

5.4 | backward direction: interpolation

5.4.1 | step back

1. evaluation was a matrix vector product, and using the k-th roots of unity allows us to simplify the product
2. interpolation is just the inverse of the DFT matrix

5.4.2 | **inverse FFT**