Discrete Fourier Transform, Fast Fourier Transform and its Application

Fourier analysis is introduced by French Mathematician Fourier when he was trying to solve some partial differential functions in 1800s. Though it is invented as a mathematical trick that mathematicians used, people later discover that it can be applied to many other aspects and is so powerful that it is now used in almost every science and engineering filed and its applications are everywhere in our daily life. In this paper I will briefly introduce the basic concepts of Fourier analysis and Fourier Transform, explain that how this trick is brought in the discrete computer world, how people develop algorithm to reduce the time complexity of this trick from O(n^2) to O(nlogn) and give a couple of applications people used the power of Fourier analysis and computers to tackle some of the problems in real life.

**Part 1. Introduction of Fourier Analysis**

I will give the following mathematical definition of Fourier Analysis:

For any periodic function f(x), we can decompose the function using a sequence of cos and sin functions. For f(x) that has a period of 2 pi, we can decompose it as the following:



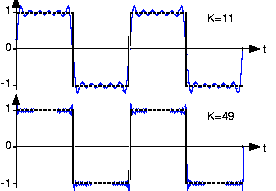
And we can calculate all the coefficients simply by multiplying the corresponding sin or cos function and then do an integration. This because that all of the sin and cos functions are all orthogonal pairwise, which means that the integration of cosnx\*cosmx=0 if n!=m and equals to 1 if n=m. Then we observer that after multiplying sinnx and do integration, we are left with b\_n only, thus finding the value of b\_n.

And this trick actually can be applied on all functions if we modify it a bit. We call this variation Fourier Transform. The Fourier transform (FT) decomposes a function of time f(t) into its constituent frequencies.(Note that it can also decompose any function f(x), it is just the decomposition is not the frequencies in our real life.).



As the above equation shows, we can transform any function f(x) to a function of w(frequency) F(w). In fact, compare to Fourier analysis, F(w) is just the different coefficients (An) shown in the last slides. It is just the coefficients become continuous now.

We will give an example to illustrate the idea of Fourier analysis intuitively, we will give a square wave, which is just a function that varies between 1 and -1 periodically. And we will calculate A\_1, A\_2 all the way to A\_11 and A\_49 in the following two graphs:



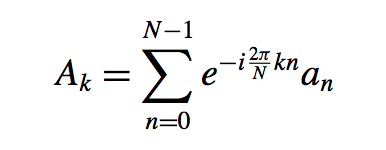
We observe that the more k we have, the more accurate approximation of the function we get. And mathematically we can show that when k goes to infinity, it will give the exactly same function, and it works with any periodic functions and work for any function if we use the tricks of Fourier Transform.

This is a good way to approximate functions as we can store the first few coefficients and then we would get a function that is really close to the original one. And this is where our first application come up. We can use it to store audio files, and it will reduce the size of the music significantly. We will give a more elaborate explanation later.

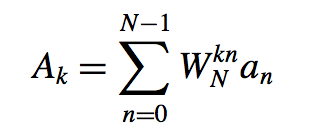
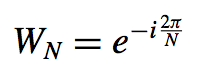
**Part 2: Discrete Fourier Transform and Fast Fourier Transform**

As the previous example has shown that we can approximate the function well with many but finite coefficients and we can also approximate integrations discretely, we now have the sufficient recipes to approximate the results of the trick discretely in our computer worlds with 0s and 1s. So now we define this process, which is formally denotated as Discrete Fourier Transform:

We can use the discrete Fourier transform, or DFT to compute fourier transform by computers. Suppose we have a periodic signal f(t) of length N. We construct sequence an for n = 0.. N-1. (Note here an =f(n)). The sequence An, the discrete Fourier transform of an is:



This is more commonly written as:

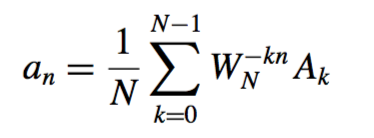
where 

Note here we use function of e rather than cos and sin as they are equivalent mathematically by Euler’s Formula. That’s why the e function is also in our first equation when introducing the Fourier Analysis.

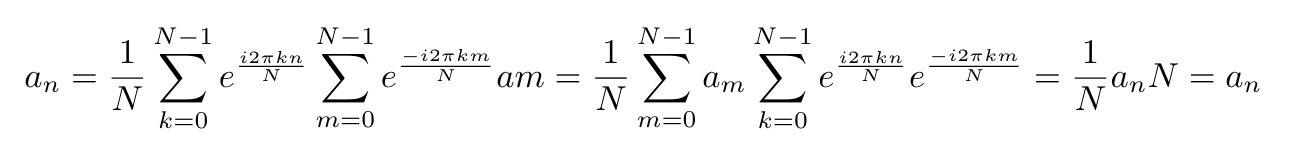
As we have explained, this is just an approximation to the original function, but it is a well enough approximation for us to solve problems.

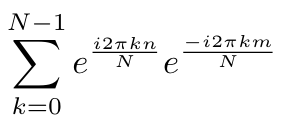
Also, given a transformed function f(w), we can also transform it back to f(x) using the same trick, just with some different normalization factors. We will also define how we do it discretely, formally denotated as the inverse discrete Fourier transform

Let sequence an be the inverse discrete Fourier transform of the sequence Ak. The formula for the inverse DFT is:

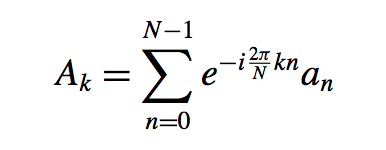


We can check it pretty straight forward:



Similar to cos and sin, we see each  as an inner product, which is multiply another sin function and do integration, and result similarly takes N only when m = n and takes 0 otherwise.

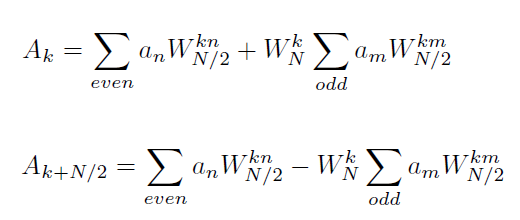
Now we have all the definitions, let’s look at how can we do this trick computationally. We begin a naive approach to the problem. Fix N, and suppose that we are given a0, a1, … , aN-1. Then the time complexity is O(N2). This is trivial according to the definition of the Fourier Transform sequence A\_N:



We observe that for every A\_k, we have to do N-1 operations and we have compute N of them. As a result, we will have O(N2) time complexity.

O(N2) is not bad, but we know that dealing with big data O(N2) is not efficient enough. Fortunately, people have developed algorithms to reduce the time complexity. The most famous one is a dynamic programming algorithm to reduce the time complexity to O(NlogN). This algorithm is referred as the Fast Fourier Transform:

The FFT algorithm is developed by Cooley & Tukey in 1965. Its main idea is “divide and conquer”, which is just another way of saying dynamic programming. The reason why we can do it is because of the following equation, which is just a variant of the definition of A\_k:



We will prove the above equation is equal to the definition of the A\_k below, but the main idea of the proof is just as W is function of e, e^(n/2)=e^(n\*1/2):

We first break the sum into even sums and odd sums:

Then we make a change of substitute let even n to be 2m and odd n to be 2m+1

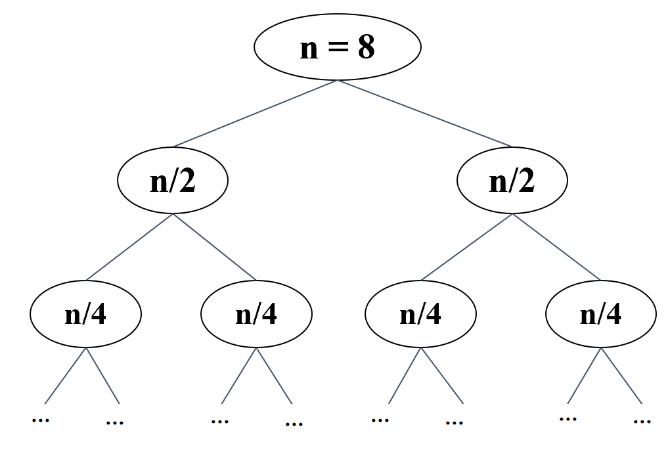
As we know that , then we know that .

Similarly,

Then we can rewrite the sum as the desired form:

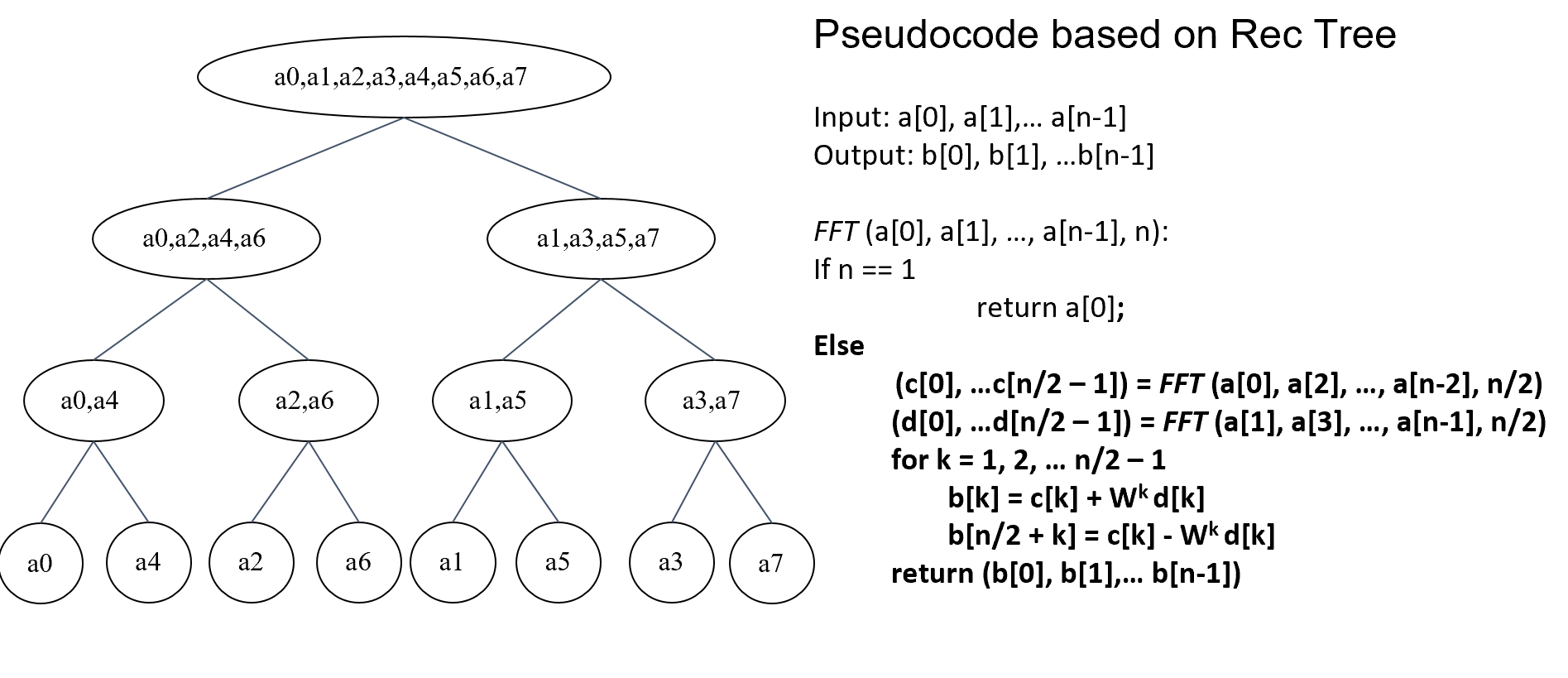
And we can derive using similar ways.

As a result, using this variant definition of A\_k, we just break this size N problem into 2 N/2 problem. Breaking it recursively, we will end up the following graph:



Obviously we will end up with logn layers, and it suffices to show that each level requires O(n) computation. This is not hard as we can see the above equation, for each sub problem size N/2^k, we will need O(N/2^k) computation with the input from next layer, and we have 2^k subproblems in each layer. Then we will have O(N) for each layer and hence O(NlogN) overall.

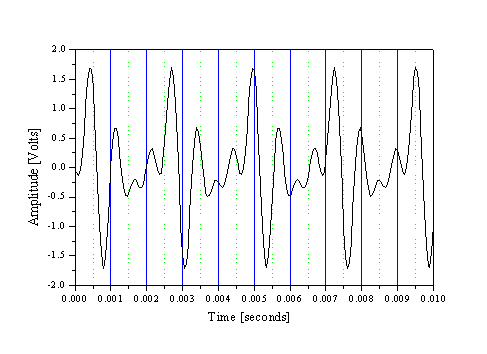
We also give the pseudo code for the problem with the illustration when the problem size is 8, and you can definitely search online to see the implementation in different languages. It might be one of the most easily found algorithm implementation online.



**Part 3: Application of Fourier Transform and FFT algorithm:**

In this section I will introduce 3 applications people used in different aspects. They are storing audio files, solving polynomial multiplication in O(NlogN) and signal processing on cosmic background radiation (CMB) to support inflation theory in field cosmology.

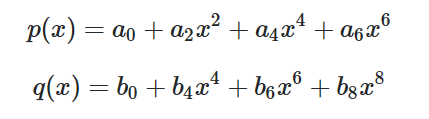
First let’s look at storing audio files. As we mentioned before, we can use Fourier Analysis to approximate function, and sound wave, as a form of smooth but complicated function, is idea to be decomposed and approximate by Fourier analysis.

given a sound wave, do fft and approximate it

Obviously, this function can hardly be expressed by some simple expression of functions, but it can be easily approximate by Fourier coefficients and can hardly be distinguished by humans when n is big enough. So we simply save coeffects so that we can produce this soundwave with these coefficients. This is actually how we save audio files by .wav format and all other formats are just compressions based on .wav format. Of course, we will have to do some other implantations and optimization to let it work. But the basic idea is just doing a FFT and storing the ns. The concepts of “sample rate” corresponds to how many n we pick. (normally we choose 48000Hz for daily use and 192000Hz for the studios and other professional usage) With the help of Fourier analysis we can make this music small enough so that we can store it or stream it. Otherwise, we might be have to go to concerts and lives every time for some good music.

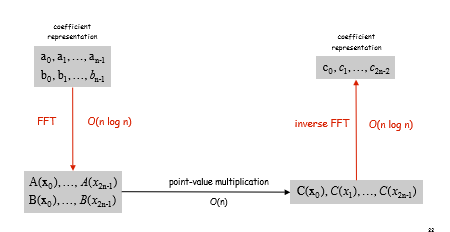
The second application is polynomial multiplication. This application is important as it can be generalized to be one of the efficient way to deal with large number multiplication. Also, with some variations, we can theoretically apply them to the “quantum computers”(if we have any) with reduced complexity O(log N) and then we can break most the crypto algorithms we used nowadays in a possible time limit. But this is beyond the scope of this paper. We will first define the problem:

Given any two polynomials with the largest degree N (below is an example of N=8)



We can compute them in O(nlogn) using FFT instead of O(n^2). The brute force method is O(n2) as we have to multiply O(n) for each term and we have O(n) terms. You can try it simply by calculating p(x)q(x) given above. And we can do it by O(n logn) using FFT. The procedure are the following:

1. Express the coefficients of f and g in terms two vectors a and b
2. Do the FFT of f and g and store them as another two vectors A and B
3. The multiplication of the new coefficients C = A\_i\*B\_i
4. Then do the inverse FFT and get vector c, which is the coefficients of the desired function

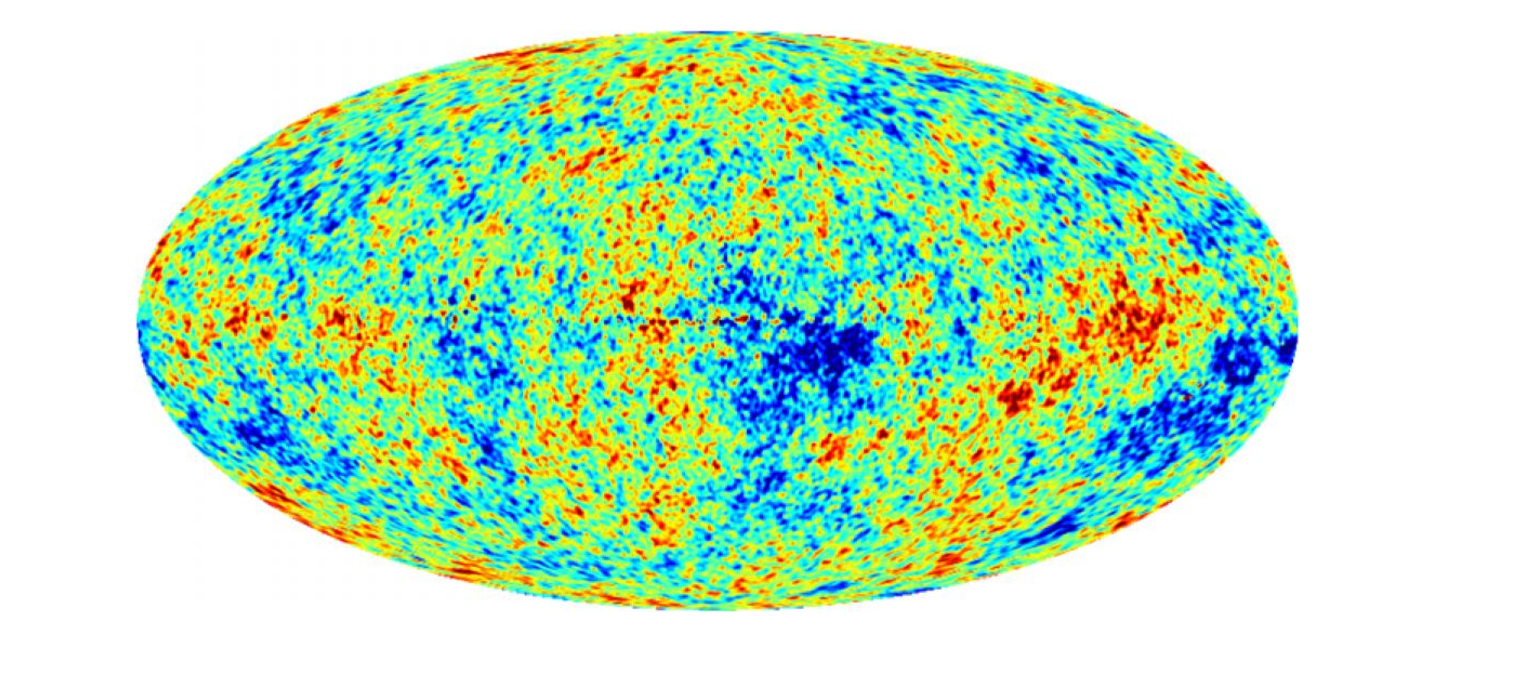


Here is math proof that such method will give out the right answer.

As the graph before shown, we take O(nlogn) to do the FFT and inverse FFT respectively and we need O(n) to do the multiplication in the fourier space when multiplying N term with the same index. Thus in total we will have O(nlog+n+nlong)=O(n logn) in total. And again, this would increase the speed significantly when dealing with polynomials with large degree.

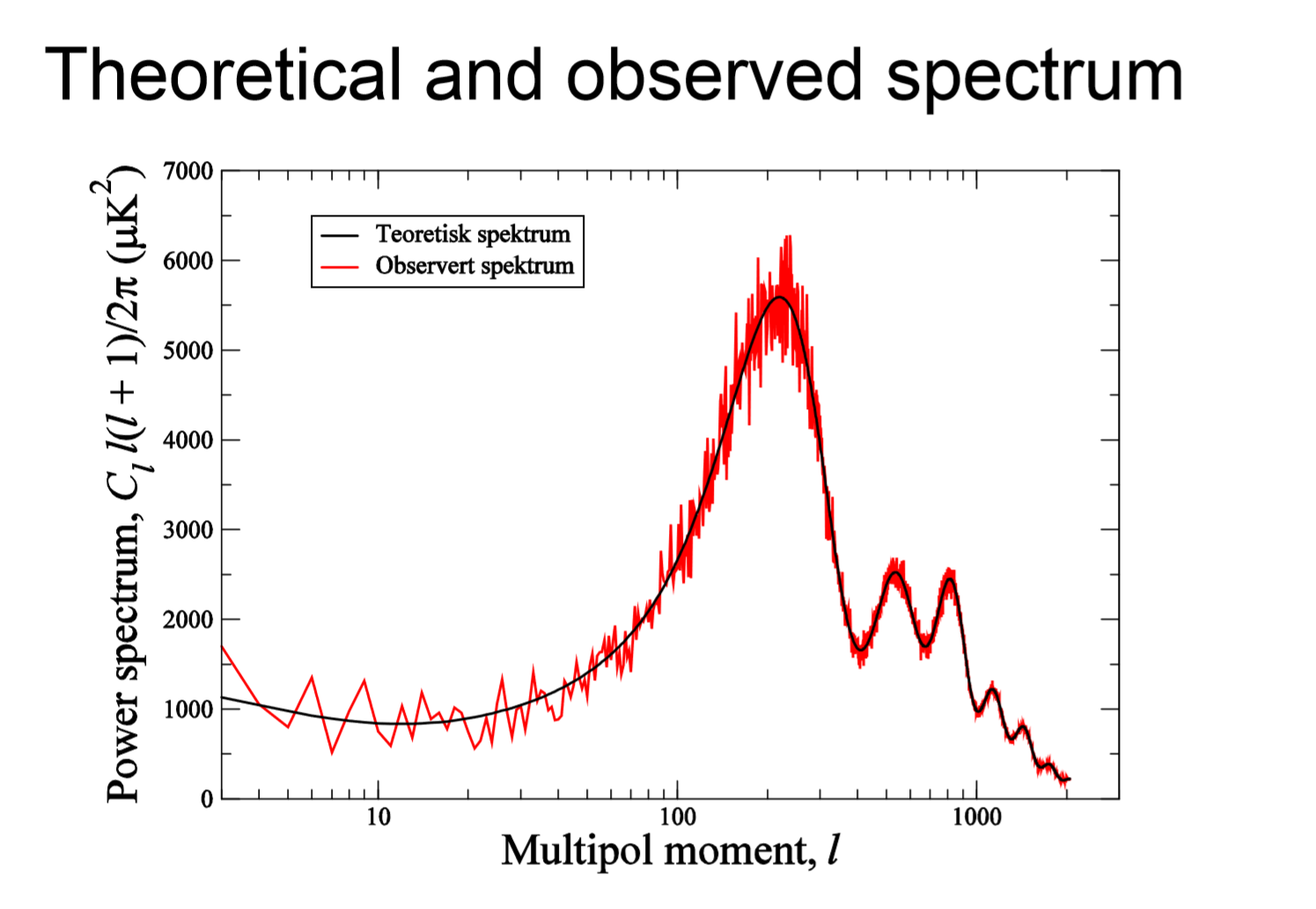
The third application is just one of the applications in the field of signal processing where Fourier is most heavily used. It is basically extracting useful information from huge data. And fortunately, with the O(nlogn) time complexity provided by FFT, we can analysis huge data in finite time and thus come out a lot of useful results. This is just a very quick and simple introductions so I apologize first for the over-simplification on the problem.

CMB stands for **cosmic microwave background (radiation).** It is the radiation as a remnant from an early stage of the universe after inflation. The CMB is the oldest and cleanest source of information in the early universe. We can detect it using just TV antenna, but it requires some fancy antennas. After detecting the signal (which is basically nothing at all), we will use some fancy data cleaning skills and we will end up with something below:



This is just the CMB temperature (which has an inverse relation with the density in that region of the universe) on a sphere, just like what on a world of the map,

We can hardly get anything useful from this graph, although we can see the different temperature, it does not seem to have any pattern. However, with the help of some decomposition (note that people are using a set of “spherical harmonics” functions instead of sin cos functions to decompose the signal in order to get meaningful results in 3D but the ideology is very similar), we end up with something below:



Apparently, we can find peaks and downs in this fast spherical harmonic transform graph and everyone now can find a unusual pattern here. And this is later explained by the cosmologists as the evidence for the inflation theory and the similar signal processing skills used on this signal has let people found lot about our early universe and led to the discovery of the dark energy and two noble prizes.