Different algorithms of FFT and IFFT

Uladzislau Zubets

December 2, 2016

Contents

1 FFT (Fast Fourier Transform):

1^O algorithm (associated to 1^O IFFT algorithm):

1. This algorithm is recursive
2. Higher memory requirements and redundancy
3. More intuitive than 2^O algorithm
4. Simply to use and implement
5. Uses even and odd variables
6. It has complex Fourier transformation
7. Requires high number discretization intervals (if frequency of discretization (fd) = frequency of signal (fs), then information is lost; if (fd = 2fs) > fs, then information is save)
8. It needs O(N^2) algebraic operations and it has also low propagation error with O(N x log N)
9. It is made by CooleyTukey FFT algorithm (in-place, divide-and-conquer: divide even and odd, conquer (calculates individually each one) fft-even and fft-odd)

2^O algorithm (associated to 1^O IFFT algorithm):

1. This algorithm is recursive
2. Better optimized (low memory requirements) than 1^O algorithm
3. Less intuitive than 1^O algorithm
4. It has complex Fourier transformation
5. More complex (difficulty) than 1^O algorithm
6. Uses even and odd variables
7. It has complex Fourier transformation
8. Requires low interval of discretization
9. It is made by CooleyTukey FFT algorithm (in-place, breadth-first, decimation-in-frequency)

Note: I advise to use this algorithm because it is faster (although 1^O algorithm is most fast) than other algorithms, it has discrete and complex Fourier transformation. It has medium optimization when it is compared with other algorithms. Uses very efficient CooleyTukey FFT algorithm. It was tested on practice and demonstrated excellent results such as itself associated IFFT.

3^O algorithm (included FFT and IFFT):

1. This algorithm is recursive
2. It has complex Fourier transformation
3. This algorithm has high complexity

4^O algorithm:

1. This algorithm is recursive
2. It is recommended to be applied to Gauss beam (variable frequency along time axis)
3. It is necessary extraordinary to be accurate when are computing frequencies
4. For higher precision is necessary to adjust parameters, to use correct window, to use non - recursive FFT
5. Very fast

6. This algorithm is adjusted to wavelets propriety like as FFTW (variable frequency along time axis)

5^O algorithm:

1. This algorithm is recursive
2. It is made by Cooley-Tukey algorithm
3. This algorithm has high complexity
4. It needs O(N^2) algebraic operations and it has also low propagation error with O(N log N)
5. It is recommended to be applied to Gauss beam (frequency impulse changing along time axis)
6. It has maximum flexibility
7. Very fast velocity
8. A complex number class meant to simplify the handling of complex numbers, at some expense of execution speed (Complex.cpp and Complex.h, although can be compiled without Complex.x)
9. A Discrete Fourier Transform routine, included for its simplicity and educational value. Very slow. Users can invoke this conversion with " ./fft$_p rocessor - d$" ($DFT.cppandDFT.h, althoughcanbecc$
10. A well-optimized Fast Fourier Transform using the Danielson-Lanzcos lemma. This routine requires that the array size be a power of 2. Notice that an array of bit-reversed pointers is created in advance of computation, an optimization that pays off during multiple conversions using the same array size (for a single conversion it has no effect)
11. Readers familiar with FFT code examples may be surprised by the relative brevity of the main conversion routine. This results from use of a Complex number class (see above) that handles some operations in a way that allows a simplification of the source, probably at the expense of execution speed

6^O algorithm (included FFT and IFFT):

1. This algorithm is recursive
2. It is made by Cooley-Tukey algorithm
3. It has complex Fourier transformation
4. This algorithm has high complexity
5. It needs O(N^2) algebraic operations and it has also low propagation error with O(N x log N)
6. Exist two functions for forward transform and two ones for inverse transform
7. Every couple consists of in-place version and version that preserves input data and outputs transform result into provided array
8. Protected section of the class has as well four functions: two functions for data pre-processing putting them into convenient order, core function for transform performing and auxiliary function for scaling the result of inverse transform
9. Inside wrapper you can find check of input parameters, then data preparation rearrangement, and transform itself

10. Rearrange function uses our mirrored mathematics to define new position for every element and swaps elements

11. While loop implements addition of 1 in mirrored manner to get target position for the element

12. Here in-place forward Fourier transform performed for signal of 1024 samples size

13. There is also Sand-Tukey algorithm that rearranges data after performing butterflies and in its case butterflies look like recursive, but mirrored to the right so that big butterflies come first and small ones do last

14. From all considerations follows that length of input data for our algorithm should be power of 2. In the case length of the input data is not power of 2 it is good idea to extend the data size to the nearest power of 2 and pad additional space with zeroes or input signal itself in a periodic manner just copy necessary part of the signal from its beginning. Padding with zeroes usually works well

7^O algorithm (included FFT and IFFT):

1. This algorithm is recursive
2. It is made by Cooley-Tukey algorithm
3. It needs O(N^2) algebraic operations and it has also low propagation error with O(N x log N)
4. It has complex Fourier transformation
5. This algorithm has high complexity
6. Makes the butterfly transform
7. Moves elements of the array as required by the iterative FFT implementation
8. Takes time O(N x log(N)) where N must be a power of 2

2 IFFT (Inverse Fast Fourier Transform):

1^O algorithm (associated to 1 and 2 FFT algorithm):

1. This algorithm is recursive
2. Higher memory requirements and redundancy
3. Simply to use and implement
4. Uses even and odd variables
5. It has complex Fourier transformation
6. Requires high number discretization intervals (if frequency of discretization (fd) = frequency of signal (fs), then information is lost; if (fd = 2fs) ¿ fs, then information is save)
7. It needs O(N^2) algebraic operations and it has also low propagation error with O(N log N)
8. It is made by CooleyTukey FFT algorithm (in-place, divide-and-conquer)