EXPERIMENT REPORT

of

Digital Signal Processing

FAST FOURIER TRANSFORM BY USING MATLAB

SUBMITTED BY

Xu Yining *2014141453202*

Professor Zhang Hong

School of Electronic and Information Engineering, Sichuan University

December, 2016

1 Fast Fourier Transform

1.1 Background of Fourier Transform

1.1.1 Fourier Transform

The Fourier transform decomposes a function of time (a *signal*) into the frequencies that make it up, in a way similar to how a musical chord can be expressed as the amplitude (or loudness) of its constituent notes. The Fourier transform of a function of time itself is a complex-valued function of frequency, whose absolute value represents the amount of that frequency present in the original function, and whose complex argument is the phase offset of the basic sinusoid in that frequency. The Fourier transform is called the *frequency domain representation* of the original signal. The term *Fourier transform* refers to both the frequency domain representation and the mathematical operation that associates the frequency domain representation to a function of time.

1.1.2 Discrete Fourier Transform

In mathematics, the *discrete Fourier transform* (DFT) converts a finite sequence of equally-spaced samples of a function into an equivalent-length sequence of equally-spaced samples of the *discrete-time Fourier transform*(DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence. An inverse DFT is a *Fourier series*, using the DTFT samples as coefficients of complex sinusoids at the corresponding DTFT frequencies. It has the same sample-values as the original input sequence.

The sequence of N complex numbers $x_0, x_1, \ldots, x_{N-1}$, is transformed into an N-periodic sequence of complex numbers:

$$X_k \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i k n}{N}}, \qquad k \in Z$$

Because of periodicity, the customary domain of k actually computed is $[0, N-1]$. That is always the case when the DFT is implemented via the Fast Fourier transform algorithm. But other common domains are $\left[-\frac{N}{2}, \frac{N}{2}-1\right]$ and $\left[-\frac{N-1}{2}, \frac{N-1}{2}\right]$ (N odd), as when the left and right halves of an FFT output sequence are swapped.

The transform is sometimes denoted by the symbol F, as in $\mathrm{X} = F\{x\} or F(x) or Fx$.

1.1.3 Fast Fourier Transform

The DFT is obtained by decomposing a sequence of values into components of different frequencies. but computing it directly from the definition is often too slow to be practical. A *fast Fourier transform* (FFT) algorithm computes the discrete Fourier

transform (DFT) of a sequence, or its inverse. Fourier analysis converts a signal from its original domain (often time or space) to a representation in the and vice versa. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, it manages to reduce the complexity of computing the DFT from $O(n^2)$, which arises if one simply applies the definition of DFT, to $O(n \log n)$, where n is the data size.

2 Fast Fourier Transform Using MATLAB

2.1 Fast Fourier Transform Algorithms

There are many different FFT algorithms involving a wide range of mathematics. The best-known FFT algorithms depend upon the factorization of *N*, but there are FFTs with$O(N \log N)$ complexity for all *N,* even for prime *N.* Many FFT algorithms only depend on the fact that $e^{-2\pi i/N}$ is an *N*th primitive root of unity, and thus can be applied to analogous transforms over any finite field, such as number-theoretic transforms. Since the inverse DFT is the same as the DFT, but with the opposite sign in the exponent and a 1/N factor, any FFT algorithm can easily be adapted for it.

2.2 Recursive Algorithm of Fast Fourier Transform

2.2.1 Definition

The recursive algorithm of *fast Fourier transform* uses a divide-and-conquer strategy, which uses the coefficients of the even-numbered and odd-numbered subscripts in $A(x)$ to define a new number of books $A^{[0]}(x)$ and$A^{[1]}(x)$:

$$A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{\frac{n}{2}-1}$$

$$A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{\frac{n}{2}-1}$$

Therefore, $A(x)$ can be expressed as follow:

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

Replace *x* to the frequency domain *w*, we can easily attain the Fourier transform.
According to the lemma, the sequence of values is not composed of n different values, but consists of $n/2$ unit complex roots, each root appear twice. Thus, the polynomials $A^{[0]}(x)$ and$A^{[1]}(x)$ with a bound of $n/2$ are recursively evaluated.
That is the most significant of recursive algorithm of fast Fourier transform.

2.2.2 MATLAB Codes (*with Notes*)

```
function Y = yn_fft1(a)
n = length(a);
if n == 1
    Y = a;
```

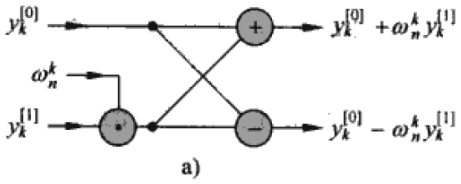
```
    return;
end
Wn = exp(1i*2*pi/n).^(0:n/2 - 1);
a0 = a(1:2:n-1);
a1 = a(2:2:n);
Y0 = yn_fft1(a0);
Y1 = yn_fft1(a1);
Y(1:n/2) = Y0 + Wn.*Y1;
% Dimension to be consistent, so multiply by point
Y(n/2 + 1:n) = Y0 - Wn.*Y1;
end
```

2.3 Iterative Algorithm for Fast Fourier Transform

2.3.1 Definition

In the recursive algorithm, we need to calculate the value of $\omega_n^k y_k^{[1]}$ twice. As common subexpressions in editing terms, we can put them in a temporary variable to improve the algorithm. In this loop, we put $\omega = \omega_n^k$ to multiply $y_k^{[1]}$. The resulting product in the variable *t* which, and then let $y_k^{[0]}$ to add t or lose *t*, this string of operations into a butterfly operation. We can assume that the input vector is arranged in a tree structure, recursively call each node of the tree, and mark the output vector. Until the vector ends with only one element.

In iterative operations, we use an array $A[0, n-1]$ to store the input vector a in the order we need to put them into the array. This operation is called bit-reversed permutation which is a binary number of a certain number of bits formed by reversing the integer.



a)

figure for Butterfly operation.

2.3.2 MATLAB Codes (*with Notes*)

```
function A = yn_fft(a)
q = ceil(log2(length(a)));
A = yn_copy(a,q);
n = length(a);
for s = 1:1:log2(n)
    m = 2^s;    % s is stand for process time
    Wm = exp(-1i*2*pi/m);
```

```
    for k = 0:m:n-1
        W = 1;
        for j = 0:1:m/2-1
            t = W*A(k+j+m/2+1);
            u = A(k+j+1);
            A(k+j+1) = u + t;
            A(k+j+m/2+1) = u - t;
            W = W * Wm;
        end
    end
end
end

function A = yn_copy(a,q)
n = length(a);
for k = 0:n-1
    b = rev(k,q);
    A(b + 1) = a(k+1);
end
% a function which is used to storage processed number
end

function y = rev(a,q)
% a function which is used to Bit-reversed permutation
t = dec2bin(a, q);
t = t(end:-1:1);
y = bin2dec(t);
end
```

3 Algorithms Comparison

3.1 Text Result

3.1.1 Text Code

```
a = [1 2 3 4 5 6 7 8];
n = 1:8;
tic
A1 = fft(a);      % result using its own function
toc
tic
A2= yn_fft(a);    % result using iterative algorithm
toc
```

```
tic
A3 = yn_fft1(a);  % result using recursive algorithm
toc
% toc is to estimate the time we take using the function
```

3.1.2 Result

```
>> A1
A1 =
  1 至 6 列
  36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -
4.0000 + 1.6569i  -4.0000 + 0.0000i  -4.0000 - 1.6569i
  7 至 8 列
  -4.0000 - 4.0000i  -4.0000 - 9.6569i

>> A2
A2 =
  1 至 6 列
  36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -
4.0000 + 1.6569i  -4.0000 + 0.0000i  -4.0000 - 1.6569i
  7 至 8 列
  -4.0000 - 4.0000i  -4.0000 - 9.6569i

>> A3
A3 =
  1 至 6 列
  36.0000 + 0.0000i  -4.0000 - 9.6569i  -4.0000 - 4.0000i  -
4.0000 - 1.6569i  -4.0000 + 0.0000i  -4.0000 + 1.6569i
  7 至 8 列
  -4.0000 + 4.0000i  -4.0000 + 9.6569i
```

As we can see in this result, both of the two algorithms can achieve fast Fourier transform.

3.2 Time of Algorithms

```
>> test
时间已过 0.003903 秒。
时间已过 0.054829 秒。
时间已过 0.011781 秒。
```

As we can see in the result, when we measured the time of each algorithms, we will find that the function contain in MATLAB takes the least time(0.003903s). And the second is the recursive algorithm(0.011781s). Moreover we are surprised to discover that the iterative algorithm takes the most of the time(0.054829s).

3.3 Comparison

In theory, the iterative algorithm is more efficient than the recursive algorithm because the iterative algorithm is processed in parallel, and the recursive algorithm is operated separately. However, we are surprised to discover that reality is not the case. due to my function, iteration algorithm bit-reversed permutation of the algorithm call the functions contained in MATLAB, *dec2bin* and *bin2dec*, these two functions of the operation speed is relatively slow, that result in slowing down the operation speed of the function itself.

4 Experience and Overview

Fast Fourier transforms are widely used for many applications in engineering, science, and mathematics. The basic ideas were popularized in 1965, but some algorithms had been derived as early as 1805. In 1994, Gilbert Strang described the FFT as "the most important numerical algorithm of our lifetime" and it was included in Top 10 Algorithms of 20th Century by the IEEE journal Computing in Science & Engineering.

Through this experiment let us a deeper understanding of the Fourier transform, and understanding of various algorithms of the fast Fourier transform of, in practice at the same time, in order to make computing more quickly, I also tried a variety of code optimization, Code editing has accumulated experience.

5 References

Cormen, Thomas H.*;* Leiserson, Charles E.*;* Rivest, Ronald L.*;* Stein, Clifford *(2001). "30. (Polynomials and the FFT)".* Introduction to Algorithms*(2 ed.). MIT Press and McGraw-Hill.* ISBN 0-262-03293-7.