An Introduction and Analysis of the Fast Fourier Transform

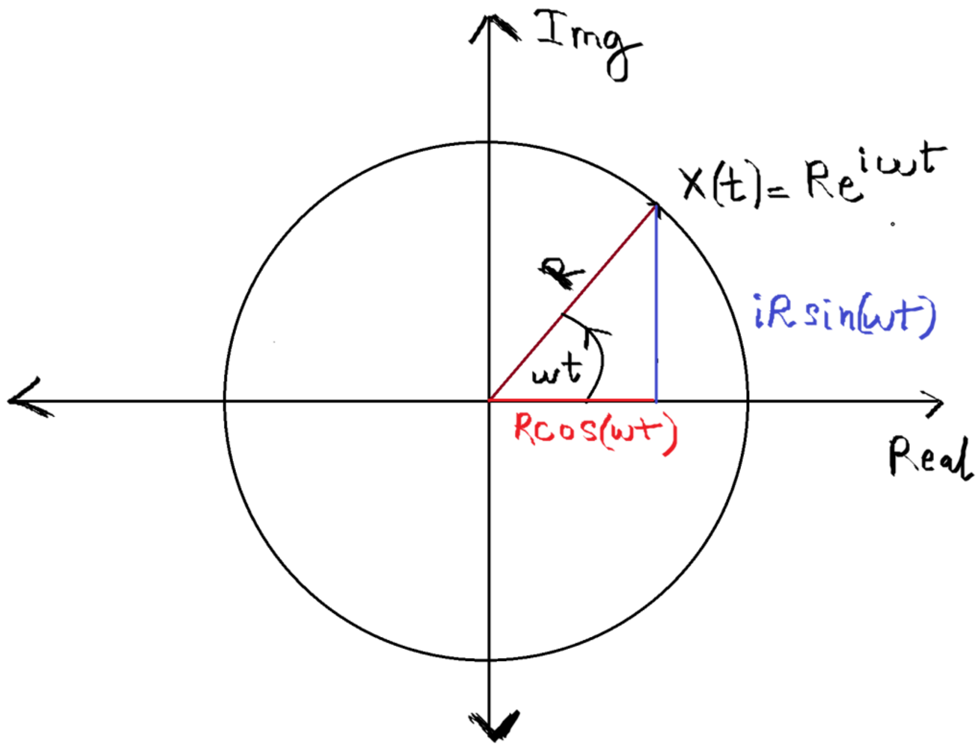
Thao Nguyen

Mentor: Professor Ron Buckmire

Outline

1. Outline of the Talk
2. Continuous Fourier Transform (CFT)
   - Visualization of the CFT
   - Mathematical description
3. Discrete Fourier Transform
   - Theory (developed from CFT)
   - Applications and an example
4. Fast Fourier Transform
   - Theory (splitting DFT into 2 recursively)
5. Time Complexity
   - Definition
   - DFT
   - Cooley-Tukey's FFT
6. Examples comparing real time complexity
   - DFT
   - FFT
7. FFT application on Solar Array data
8. Conclusion

Continuous Fourier Transform



In the Polar coordinate with $r = R$ and $\theta = wt$

$$x(t) = Re^{iwt}$$

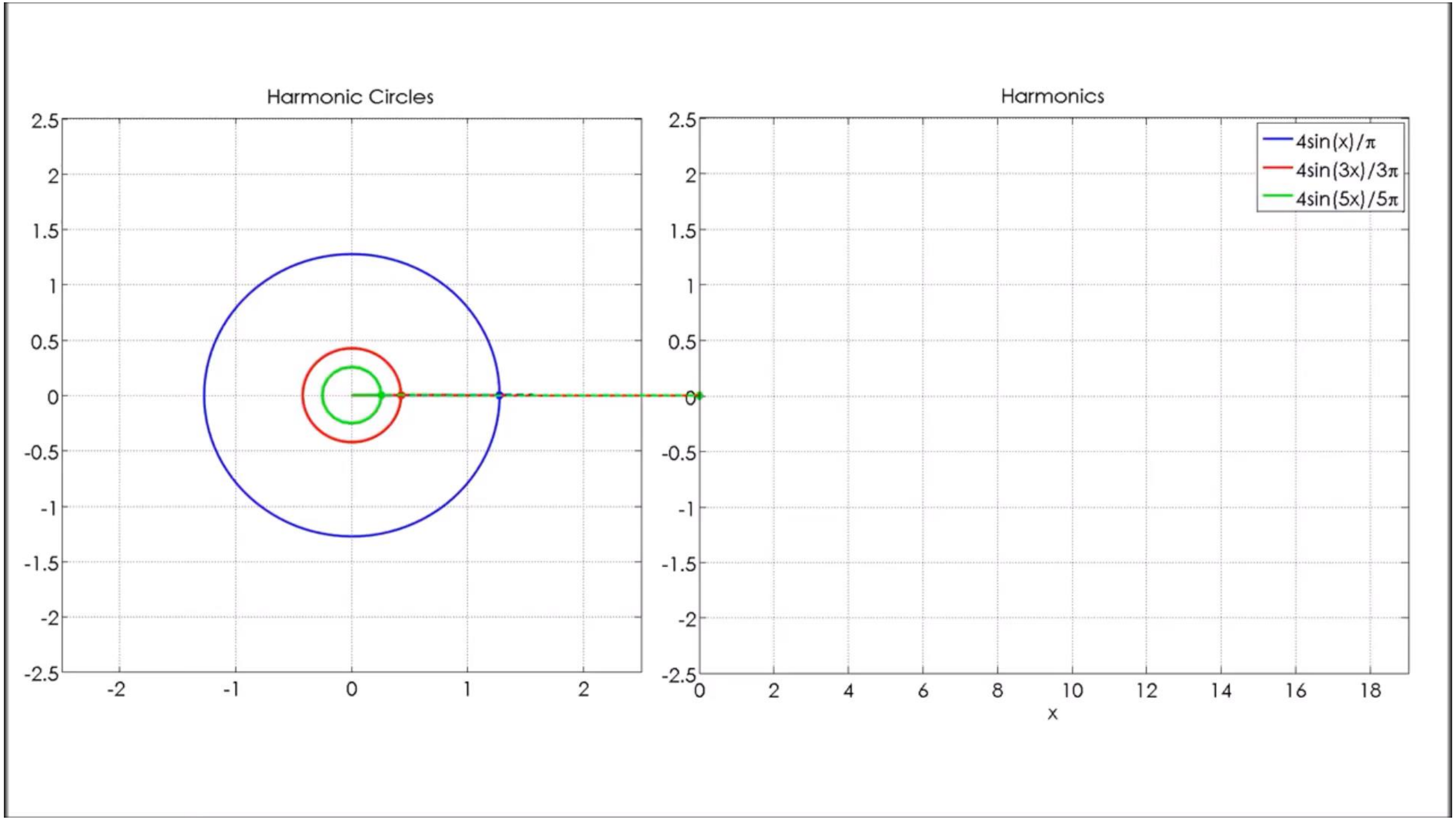
In the Cartesian coordinate

$$x(t) = Rcos(wt) + iRsin(wt)$$

Continuous Fourier Transform

The position *x* of time *t* is a periodic function of time.

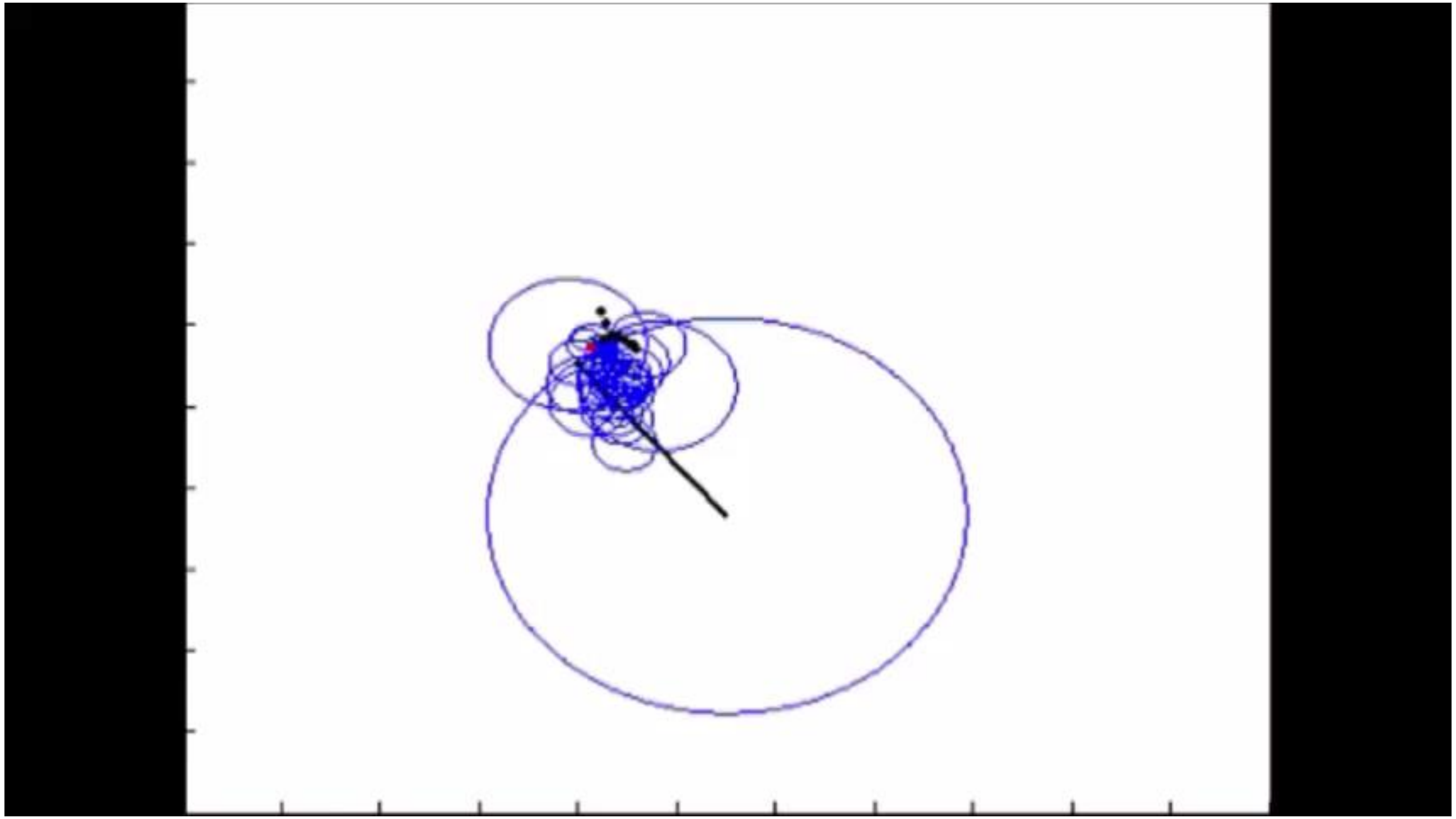
(https://youtu.be/LznjC4Lo7lE?t=28s)



Continuous Fourier Transform

If we add enough circles with different sizes and given appropriate angular frequency for each circle, we can create any shape [5].

(https://www.youtube.com/watch?v=QVuU2YCwHjw)



Continuous Fourier Transform

The path *x(t)* has *n* different circles each with unique angular frequency w_j, and radius R_j, can be described on the complex plane as a sum of all the circles:

$$x(t) = \sum_{j=1}^{n} R_j e^{i w_j t} \quad (1)$$

For a non-periodic function *x(t)*, we can assume that its period is approaching infinity, so it is represented by an infinite number of circles.

$$x(t) = \int_{-\infty}^{\infty} R(w) e^{iwt} dw \quad (2)$$

$$R(w) = \int_{-\infty}^{\infty} x(t) e^{-iwt} dt \quad (3)$$

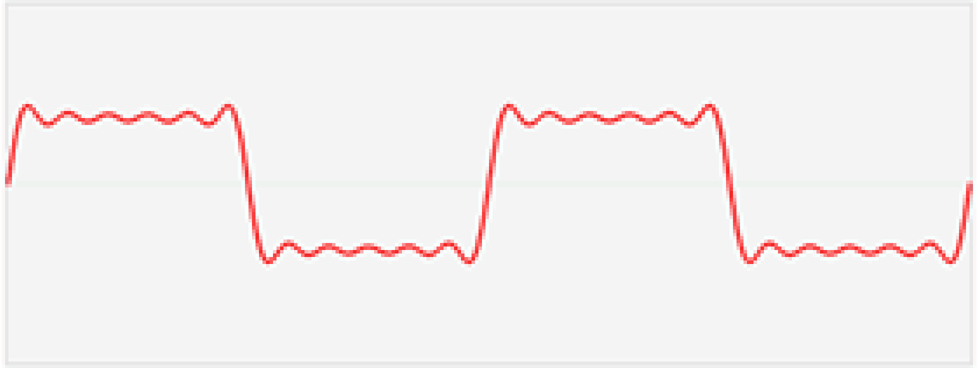
The function of angular frequency *R(w)* is the continuous Fourier transform of the function of time *x(t)*.

Fourier Transform decomposes any function into periodic functions [4].

Continuous Fourier Transform

$$x(t) = \int_{-\infty}^{\infty} R(w) e^{iwt} dw$$

Let $x(t) = S_6(\mathrm{x})$, and $R(w) = S(f)$



The Gif was originally created by a Wikipedia user, can be found at
https://en.wikipedia.org/wiki/File:Fourier_transform_time_and_frequency_domains_(small).gif

Discrete Fourier Transform (DFT)

Discretizing the continuous Fourier Transform:

- Let $x(T_n)$ be the n^{th} element of the finite sequence $\{x_N\}$, with T as the discrete sampling interval between each data point. So $T_n = T \times n$.
- $X(w_k)$ is the DFT of $x(T_n)$ [1].

$$X(w_k) = \sum_{n=0}^{N-1} x(T_n)e^{-iw_kT_n} \quad (4)$$

with the k^{th} angular frequency $w_k = \frac{2\pi k}{NT}$

- Substitute $w_k = \frac{2\pi k}{NT}$, and $T_n = T \times n$, we have:

$$X(w_k) = \sum_{n=0}^{N-1} x(T_n)e^{-\frac{2\pi i n k}{N}} \quad (5)$$

For $0 \leq k < N$

Discrete Fourier Transform (DFT) Application

"This is the most important numerical algorithm of our lifetime..."
– Gilbert Strang (Chauvenet Prize 1977 Recipient)

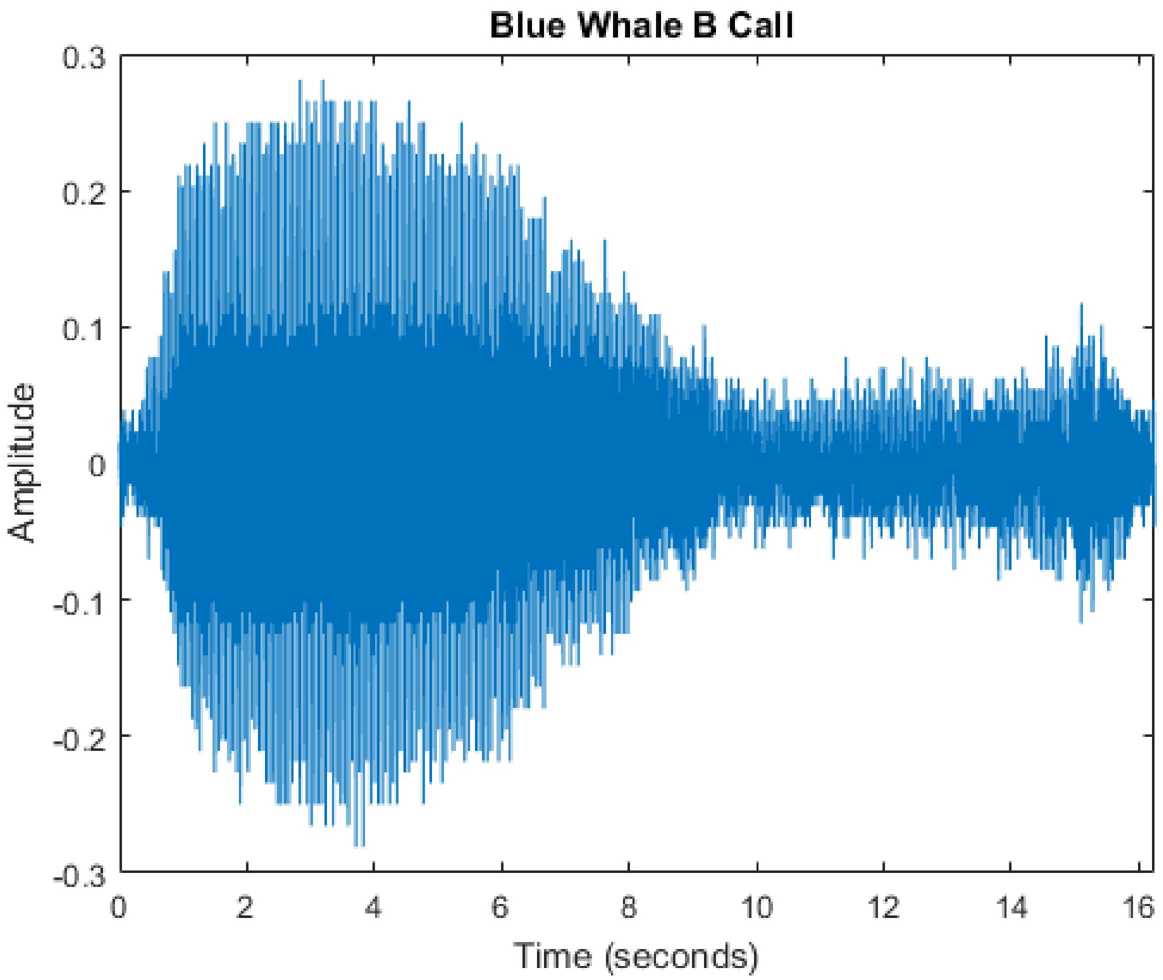
- Digital signal processing: spectral analysis of signal (human speech and hearing), Frequency Response of Systems (system analysis in frequency domain), Convolution via the Frequency Domain [8].

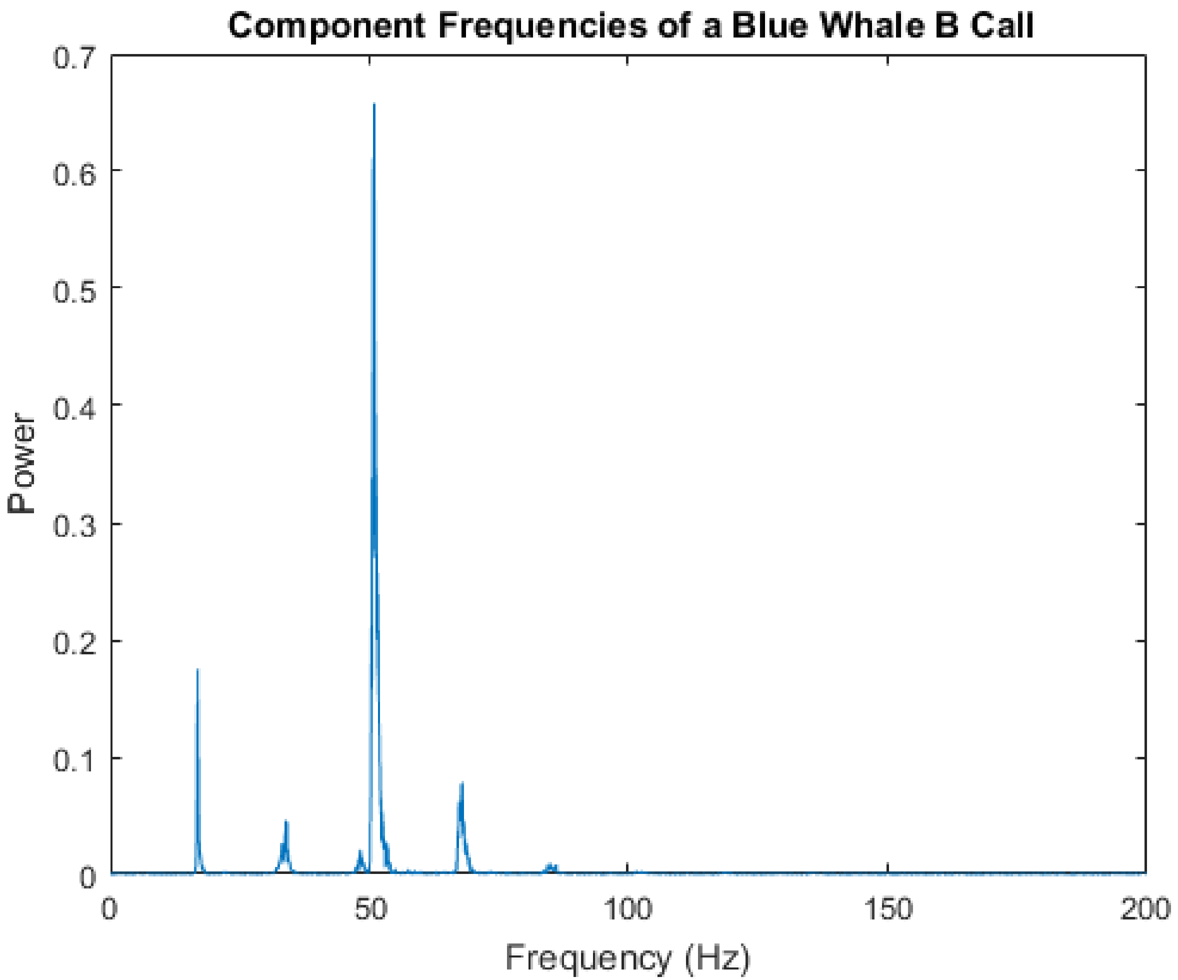
- Image processing: image analysis, image filtering, image reconstruction and image compression.

- Solving partial differential equation

Discrete Fourier Transform (DFT) Application

<u>Digital signal processing:</u> an analysis of a Blue Whale call [10]





Fast Fourier Transform (FFT)

- From (5) on page 8

$$X(w_k) = \sum_{n=0}^{N-1} x(T_n) e^{-\frac{2\pi ink}{N}}, \qquad 0 \leq k < N$$

- Let the DFT of the $x(T_n)$ data points be written as the sum of an even-indices *n=2m* as E_k and odd indices *n=2m+1* as O_k.

$$E_k = \sum_{m=0}^{N/2-1} x(T_{2m}) e^{-\frac{2\pi ik(2m)}{N}} \; and \; O_k = \sum_{m=0}^{N/2-1} x(T_{2m+1}) e^{\frac{-2\pi ik(2m)}{N}} \quad (6)$$

$$X(w_k) = \sum_{m=0}^{N/2-1} x(T_{2m}) e^{-\frac{2\pi ik(2m)}{N}} + \sum_{m=0}^{N/2-1} x(T_{2m+1}) e^{\frac{-2\pi ik(2m+1)}{N}} \quad (7)$$

$$= E_k + e^{\frac{-i2\pi k}{N}} O_k \quad (8)$$

Fast Fourier Transform (FFT)

$$X(w_k) = E_k + e^{\frac{-i2\pi k}{N}} O_k \qquad (8)$$

Since the even and odd indices DFT is periodic over *N/2*, so $E_k = E_{k\pm\frac{N}{2}}$ and $O_k = O_{k\pm\frac{N}{2}}$

We can write $X(w_k)$ such that *k* can be reduced in half, from 0 to N/2:

$$X(w_k) = E_k + e^{\frac{-i2\pi k}{N}} O_k \qquad \text{for } 0 \leq k < \frac{N}{2} \qquad (9)$$

$$X(w_k) = E_{k-\frac{N}{2}} + e^{\frac{-i2\pi k}{N}} O_{k-\frac{N}{2}} \qquad \text{for } \frac{N}{2} \leq k < N \qquad (10)$$

Also since $e^{\frac{-i2\pi(k+\frac{N}{2})}{N}} = e^{\frac{-i2\pi k}{N} - i\pi} = e^{-i\pi} e^{\frac{-i2\pi k}{N}} = -e^{\frac{-i2\pi k}{N}}$

$$X(w_{k+\frac{N}{2}}) = E_{k+\frac{N}{2}} + e^{\frac{-i2\pi\left(k+\frac{N}{2}\right)}{N}} O_{k+\frac{N}{2}} = E_k - e^{\frac{-i2\pi k}{N}} O_k \qquad (11)$$

Fast Fourier Transform (FFT)

The FFT algorithm recursively break the DFT into even and odd indices DFT E_k and O_k then calculate these smaller DFT

$$X(w_k) = E_k + e^{\frac{-i2\pi k}{N}} O_k \tag{9,10}$$

$$X(w_{k+\frac{N}{2}}) = E_k - e^{\frac{-i2\pi k}{N}} O_k \tag{11}$$

$$\text{with } 0 \leq k < \frac{N}{2}$$

Time Complexity

- "Complexity can be viewed as the maximum number of primitive operations that a program may execute. Regular operations are single additions, multiplications, assignments etc. We may leave some operations uncounted and concentrate on those that are performed the largest number of times" [2].
- Time complexity can be described in Big-O notation.

O(1): It takes the algorithm the same amount of time to compute, with different variables.

```
int a=1;
X= a+a;
```

O(N): The computation time depend linearly on variable N.

```
For (i=0, i<N, i++)
Print i;
```

O(N^2): The computation time depend on the quadratic of N.

```
For (i=0, i<N,i++)
    For(j=10, j<N+10,j++)
        print i+j;
```

O($\log_2 N$) : The computation time started with N, then get cut in half for each iteration loop

```
x=N;
Do{
   X=x/2;
      } while (x>0)
```

Time complexity of DFT (Matlab)

function output = dft(input)	+O(1)
t1= now;	+O(1)
N = length(input);	+O(1)
output = zeros(size(input));	+O(1)
for k = 0 : N - 1	+O(*N*)
s = 0;	*O(1)
for t = 0 : N - 1	+O(*N*)
s = s + input(t + 1) * exp(-2i *pi*t * k / N);	*O(1)
end	+O(1)
output(k + 1) = s;	+O(1)
end	+O(1)
t2 = now;	=O(1+1+1+1+*N**(1+*N**1+1)+1+1)
disp(t2-t1);	=O($N^2 + constant$)
end	

$$=O(N^2)$$

Code taken from http://www.nayuki.io/page/how-to-implement-the-discrete-fourier-transform

Time complexity of FFT (C++)

```
void fft(CArray& x)
{const size_t N = x.size();                       +O(1)
if (N <= 1)                                       +O(1)
   return;                                              *O(1)

CArray even = x[std::slice(0, N/2, 2)];           +O(1)
CArray  odd = x[std::slice(1, N/2, 2)];           +O(1)
fft(even);                                        +O(N log_2 N)
fft(odd);

for (size_t k = 0; k < N/2; ++k)                  +O(N)
   {   Complex t = std::polar(1.0, -2 * PI * k / N) * odd[k];   *O(1)
       x[k    ]      = even[k] + t;                     +O(1)
       x[k+N/2] = even[k] - t;                          +O(1)
   }
}
```

$$=O(N\log_2 N)$$

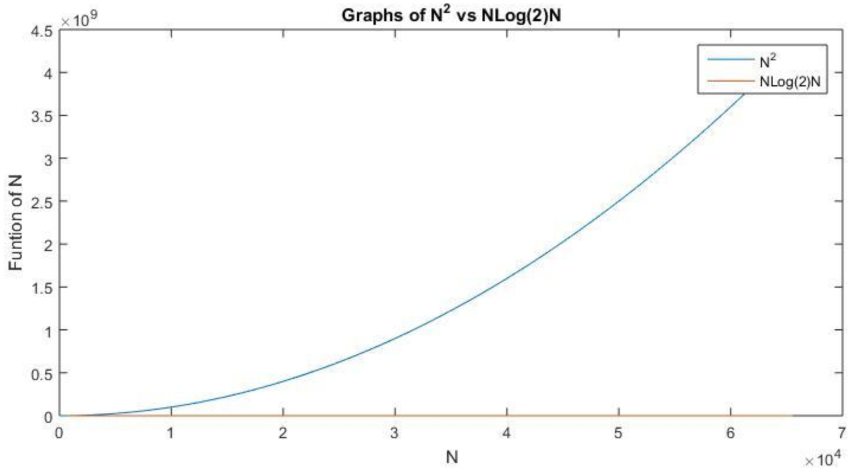
Code taken from http://rosettacode.org/wiki/Fast_Fourier_transform#C.2B.2B

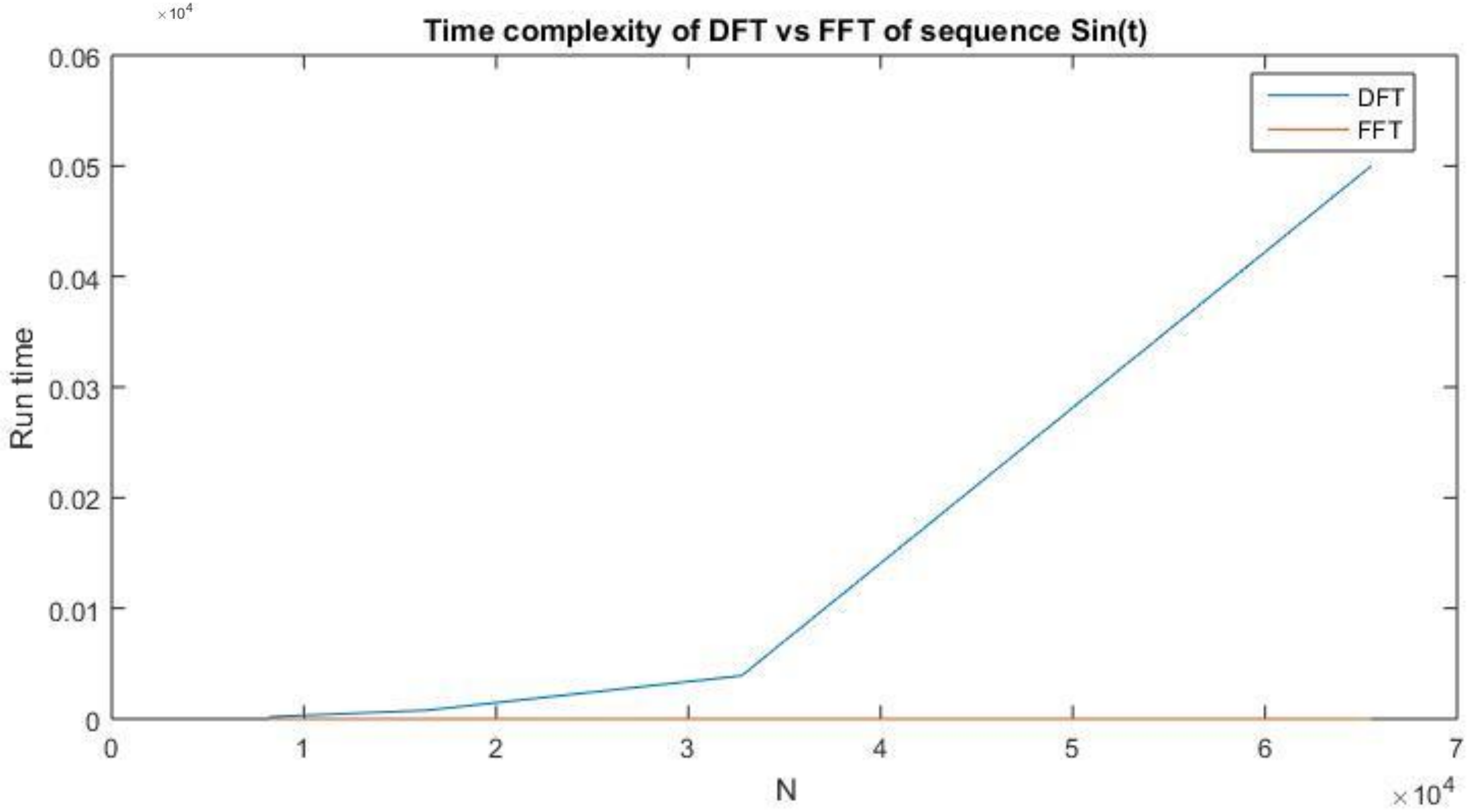
Time complexity of DFT vs. FFT

- Let $N = 2^a$, with a $\in \{13, 14, 15, 16\}$
- The time interval is from -100 to 100 with sample $t=200/N$
- The original sequence $x(t_n)$ = Sin(t_n), with n is from 0 to N-1 and $t_n = nt$

a	N $=2^a$	Run time DFT	Run time FFT
13	8192	1.88×10^{-4}	0
14	16384	7.80×10^{-4}	4.63×10^{-8}
15	32768	39×10^{-4}	4.62×10^{-8}
16	65536	50×10^{-4}	9.27×10^{-8}

Time complexity of DFT vs. FFT



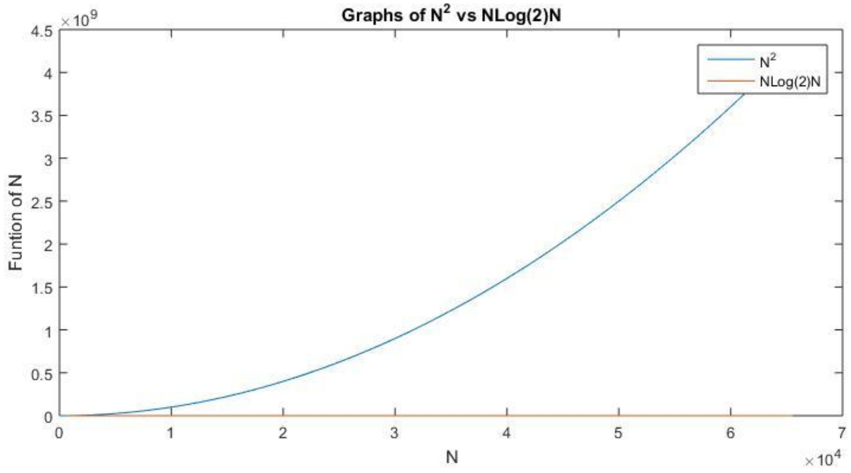


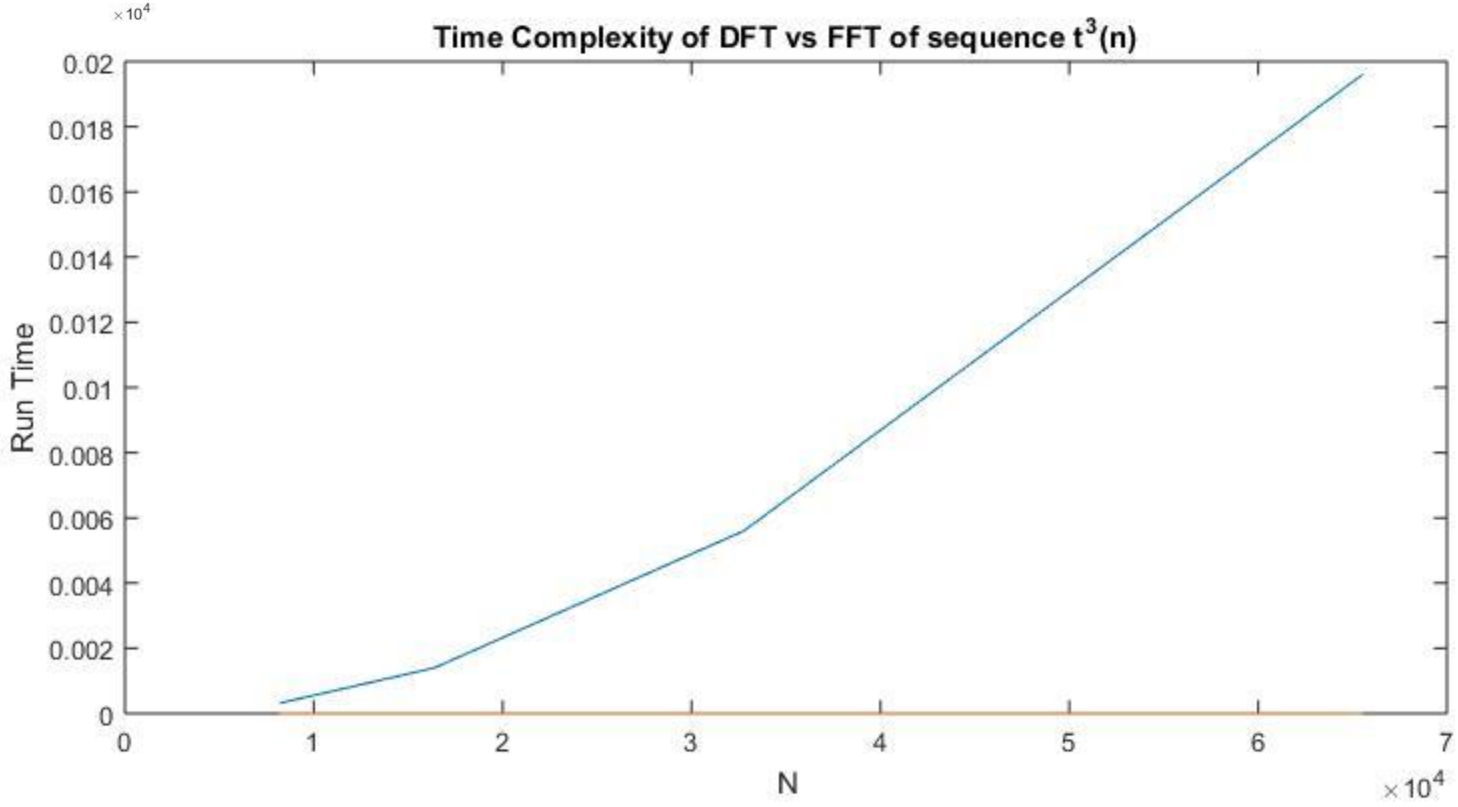
Time complexity of DFT vs. FFT

- Let $N = 2^a$, with a $\in \{13, 14, 15, 16\}$
- The time interval is from -100 to 100 with sample t=200/N
- The original sequence $x(t_n) = t_n^3$, with n is from 0 to N-1 and $t_n = nt$

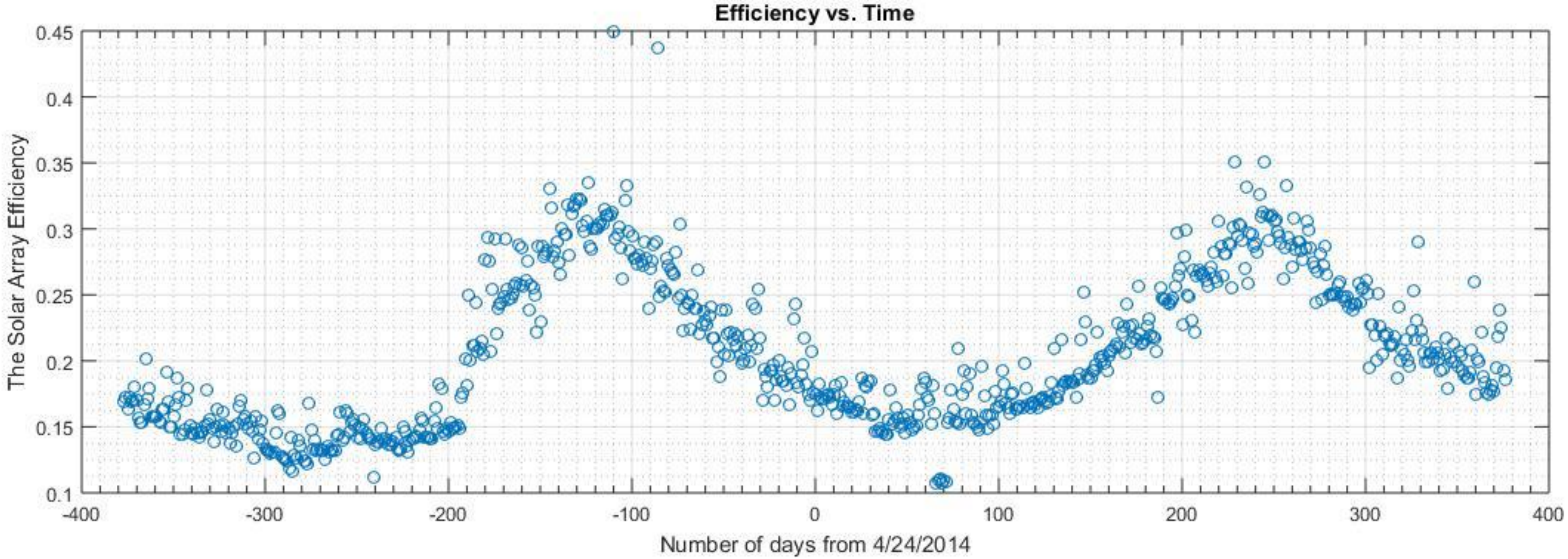
a	$N = 2^a$	Run time DFT	Run time FFT
13	8192	3.16×10^{-4}	0
14	16384	14×10^{-4}	0
15	32768	56×10^{-4}	1.15×10^{-8}
16	65536	196×10^{-4}	2.32×10^{-8}

Time complexity of DFT vs. FFT



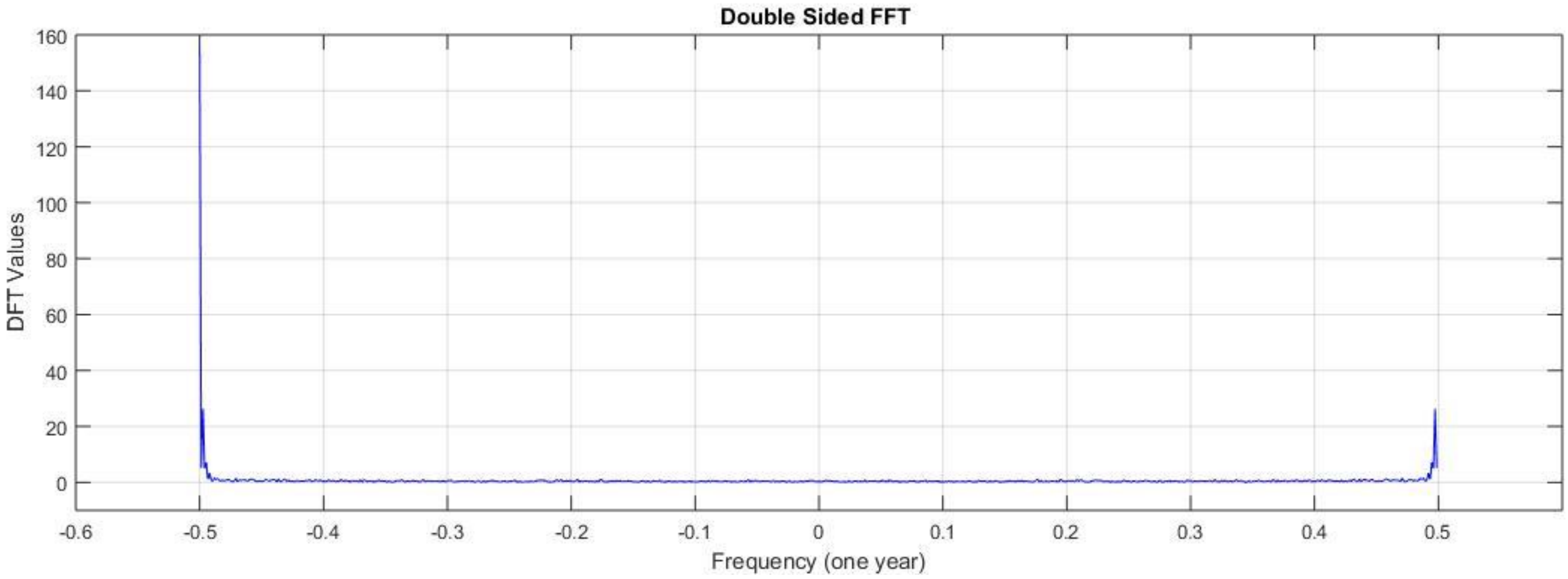


Application of FFT on Solar Array data



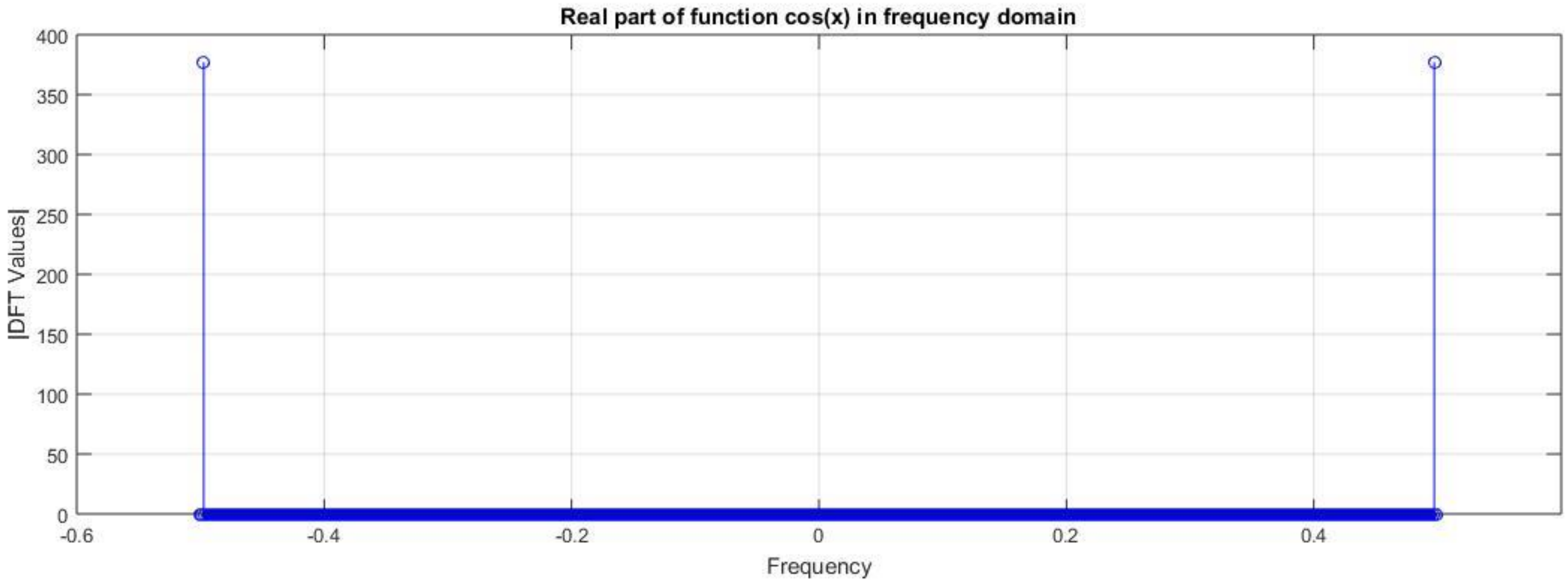
The discrete data of efficiency as a function of time looks like a periodic function.

Application of FFT on Solar Array data



The DFT of the efficiency confirms our observation that the data has a period of a year/ annually.

Application of FFT on Solar Array data



The DFT of the efficiency is similar in shape with the DFT of function Cos(x)

Conclusion

- We showed that the periodic signal of time can be represent as other primitive periodic function
- The application of the DFT in mathematics and engineering is very important as demonstrated through an example: whale signal.
- Since the DFT algorithm has a time complexity O(N^2), it is very time consuming for processing large amount of data.
- The FFT algorithm gives the same result as the DFT much faster, but with time complexity O($N\log_2 N$), allowing the run time for large amount of data to be more reasonable.

References

1. A. V. Anand, “A Brief study of Discrete and Fast Fourier Transform.”
2. Codility Ltd. https://codility.com/media/train/1-TimeComplexity.pdf.
3. D. Morin, “Fourier Analysis,” in unpublished, ch. 3. 2009.
4. F. A. Farris, “Wheels on Wheels-Surprisingly Symmetry,” in *Mathematics Magazine*, vol. 69, No.3, June 1996.
5. N. R. Hanson, “The Mathematical Power of Epicyclical Astronomy.” in *Isis*, Vol. 51, No. 2 (Jun., 1960), pp. 150-158, University of Chicago Press, September 2011. http://www.u.arizona.edu/~aversa/scholastic/Mathematical%20Power%20of%20Epicyclical%20Astronomy%20(Hanson).pdf
6. J. W. Cooley, J. W. Tukey, “An Algorithm for the Machine Calculation of Complex Fourier Series,” 1965.
7. S. G. Johnson and M. Frigo, "Implementing FFTs in practice," in *Fast Fourier Transforms* (C. S. Burrus, ed.), ch. 11, Rice University, Houston TX: Connexions, September 2008.
8. S. W. Smith, “Applications of the DFT,” in *The Scientist and Engineer's Guide to Digital Signal Processing*, ch. 9, California Technical Publishing, 1997-2011.
9. Strang, Gilbert, *"Wavelets," American Scientist* ***82*** *(3): 253*S.
10. The MathWorks, Inc, “Fast Fourier Transform (FFT).” http://www.mathworks.com/help/matlab/math/fast-fourier-transform-fft.html.