Fourier Transform for Option Pricing *(First draft – 2015Jul, updated – 2018Oct)*

Option pricing can be done by :
- solving BSPDE analytically (using heat equation)
- solving BSPDE numerically (using finite difference)
- finding risk neutral pricing analytically
- finding risk neutral pricing numerically (using binomial tree)
- finding risk neutral pricing numerically (using Monte Carlo)

Fourier transform can be used in option pricing via different approaches :
- solving BSPDE analytically using differentiation property of Fourier transform
- finding risk neutral pricing analytically using characteristic function, like Bakshi Madan
- finding risk neutral pricing numerically using characteristic function, like Carl Madan
- finding risk neutral pricing numerically using binomial tree and circular convolution

This document will cover the last approach only. Let's go through discrete Fourier transform DFT, not DTFT, first. complex plane is very good for describing clockwise or anticlockwise rotation aound unit circle in 2 dimensional space. The product of two complex numbers A and B can be regarded as scaling A by *mag(B)* and rotating A by *arg(B)* :

$$
\begin{aligned}
A &= r_A(\cos\vartheta_A + i\sin\vartheta_A) = r_A e^{i\vartheta_A} && \text{please read "Complex analysis.doc"}\\
B &= r_B(\cos\vartheta_B + i\sin\vartheta_B) = r_B e^{i\vartheta_B}\\
AB &= r_A r_B(\cos\vartheta_A + i\sin\vartheta_A)(\cos\vartheta_B + i\sin\vartheta_B)\\
&= r_A r_B(\cos\vartheta_A\cos\vartheta_B - \sin\vartheta_A\sin\vartheta_B + i\cos\vartheta_A\sin\vartheta_B + i\sin\vartheta_A\cos\vartheta_B)\\
&= r_A r_B(\cos(\vartheta_A+\vartheta_B) + i\sin(\vartheta_A+\vartheta_B))\\
&= r_A r_B e^{i(\vartheta_A+\vartheta_B)}
\end{aligned}
$$

Now, let's consider the N^{th} root of unity. There are N complex roots, all lying uniformly on the unit circle, like evenly spaced spokes of a wheel, i.e. equation $x^N = 1$ has N roots including $(e^{i2\pi/N})$, $(e^{i2\pi/N})^2$, $(e^{i2\pi/N})^3$, …, $(e^{i2\pi/N})^{N-1}$ and $(e^{i2\pi/N})^N = 1$, each of them can be generated from the previous one by anticlockwise rotation through an angle of $2\pi/N$ radians. Let's define :

$$
\begin{aligned}
& z_N = e^{i(2\pi/N)}\\
\text{then}\quad & z_N^n = e^{i(2\pi/N)n} && \forall n \in [1, N]\text{, they are } N \text{ roots of unity}\\
& z_N^n = z_N^{n \bmod N}\\
& (z_N^n)^k = (z_N^{n \bmod N})^k && \text{equation 1}
\end{aligned}
$$

If we construct a complex plane, with horizontal axis as *Real axis*, having positive values on RHS, and vertical axis as *Imaginary axis*, having positive values on the top, hence *3 o'clock* direction denotes positive real numbers, then we have :

conjugate represents reflection along x axis : $z_N^{-n} = \overline{z_N^n}$

sequence	$seq(z)$	=	$\{z_N^0, z_N^1, z_N^2, ..., z_N^{N-2}, z_N^{N-1}\}$	*anticlockwise rotation*	*(iter++)*
sequence	$rev(seq(z))$	=	$\{z_N^0, z_N^{-1}, z_N^{-2}, ..., z_N^{-(N-2)}, z_N^{-(N-1)}\}$	*clockwise rotation*	*(iter--)*
		=	$\{z_N^0, z_N^{N-1}, z_N^{N-2}, ..., z_N^2, z_N^1\}$	*clockwise rotation too*	
sequence	$seq(z^k)$	=	$\{z_N^{k\times0}, z_N^{k\times1}, z_N^{k\times2}, ..., z_N^{k\times(N-2)}, z_N^{k\times(N-1)}\}$	*anticlockwise rotation*	*(iter+=k)*
sequence	$rev(seq(z^k))$	=	$\{z_N^{k\times0}, z_N^{-k\times1}, z_N^{-k\times2}, ..., z_N^{-k\times(N-2)}, z_N^{-k\times(N-1)}\}$	*clockwise rotation*	*(iter-=k)*

Sequence $seq(z^k)$ implies that we can iterate through all spokes of the unit wheel with any *iterating interval k*, starting from *spoke 0*, i.e. the spoke pointing to *3 o'clock* direction, such that we can visit each spoke *once and only once*, right before we stop iterating when we return the starting point again. Sum of the sequence, i.e. sum of all spokes, is zero. Just imagine that they form a vector set of forces radiating from the same centre and cancel each other.

$$
\begin{aligned}
\textstyle\sum_{n=0}^{N-1} z_N^n &= (1-z_N^N)/(1-z_N) = (1-1)/(1-z_N) = 0 && && \text{since } \textstyle\sum_{n=0}^{N-1} ar^n = a(1-r^N)/(1-r)\\
\textstyle\sum_{n=0}^{N-1} z_N^{kn} &= (1-z_N^{kN})/(1-z_N^k) = (1-1)/(1-z_N^k) = 0 && \text{if } k \neq 0, \pm N, \pm 2N, \pm 3N, ... && \text{equation 2a}\\
\textstyle\sum_{n=0}^{N-1} z_N^{kn} &= 1+1+1+...+1 = N && \text{if } k = 0, \pm N, \pm 2N, \pm 3N, ... && \text{equation 2b}
\end{aligned}
$$

Discrete Fourier Transform (DFT)

Discrete Fourier transform of a complex sequence $seq(a_n) = \{a_0, a_1, a_2, \ldots, a_{N-1}\}$ is defined as another complex sequence $seq(A_k) = \{A_0, A_1, A_2, \ldots, A_{N-1}\}$, such that each element $A_k \; \forall k \in [0,N-1]$ is the normalised dot product between $seq(a_n)$ and $seq(z_N^{-k})$, whereas $seq(z_N^{-k})$ is the clockwise sequence of the N^{th} *root of unity* with iterating interval k, i.e. $seq(z_N^{-k}) = \{(z_N^{-k})^0, (z_N^{-k})^1, (z_N^{-k})^2, \ldots, (z_N^{-k})^{N-1}\}$. Please note that : *(1)* unlike $seq(z_N^{-k})$, $seq(a_n)$ does not necessarily lie on the unit circle, the elements of $seq(a_n)$ are not necessarily ordered according to their arguments, they are not necessarily evenly spaced neither, *(2)* the only limitation on $seq(a_n)$ is that its size should be N, therefore we can define $a_n = a_{n \bmod N}, \forall n > N$, and $a_n = a_{n+N} = a_{n+2N} = \ldots \; \forall n < 0$, finally *(3)* we use $n\&m$ as the time index while k as the frequency index. Similarly, inverse discrete Fourier transform of $seq(A_k)$ is defined as the complex sequence $seq(b_n) = \{b_0, b_1, b_2, \ldots, b_{N-1}\}$, such that each element b_n is the normalised dot product between $seq(A_k)$ and $seq(z_N^n)$, whereas $seq(z_N^n)$ is the anticlockwise sequence of the N^{th} *root of unity* with iterating interval n.

$$A_k = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} a_n z_N^{-kn} \qquad \forall k \in [0, N-1] \qquad \textit{known as DFT} \qquad \textit{and we write } seq(A_k) = F(seq(a_n))$$

$$b_n = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} A_k z_N^{kn} \qquad \forall n \in [0, N-1] \qquad \textit{known as IDFT} \qquad \textit{and we write } seq(b_n) = F^{-1}(seq(A_k))$$

We can prove that $seq(b_n)$ is equivalent to $seq(a_n)$, i.e. $seq(a_n) = seq(b_n) = F^{-1}(seq(A_k)) = F^{-1}(F(seq(a_n)))$.

$$\begin{aligned}
b_n &= \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} A_k z_N^{kn} \\
&= \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}\left(\frac{1}{\sqrt{N}}\sum_{m=0}^{N-1} a_m z_N^{-km}\right) z_N^{kn} \\
&= \frac{1}{N}\sum_{m=0}^{N-1} a_m \left(\sum_{k=0}^{N-1} z_N^{k(n-m)}\right) && \textit{recall } \sum_{k=0}^{N-1} z_N^{k(n-m)} = \begin{cases} 0 & \textit{if } m \neq n \\ N & \textit{if } m = n \end{cases} \\
&= \frac{1}{N}\left(\underbrace{a_0 \sum_{k=0}^{N-1} z_N^{k(n-0)}}_{0} + \ldots + a_n \underbrace{\sum_{k=0}^{N-1} z_N^{k(n-n)}}_{N} + \ldots + a_{N-1} \underbrace{\sum_{k=0}^{N-1} z_N^{k(n-(N-1))}}_{0} \right) && = \; a_n
\end{aligned}$$

Remark 1 – Shifting spokes

DFT is unchanged if we shift $seq(a_n)$ by m spokes and shift $seq(z_N^{-k})$ by $m \times k$ spokes :

$$\begin{aligned}
\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} a_{n+m} z_N^{-k(n+m)} &= \frac{1}{\sqrt{N}}\sum_{l=m}^{N-1+m} a_l z_N^{-kl} && \textit{put } l = n+m \\
&= \frac{1}{\sqrt{N}}\left(\sum_{l=m}^{N-1} a_l z_N^{-kl} + \sum_{l=N}^{N-1+m} a_l z_N^{-kl}\right) \\
&= \frac{1}{\sqrt{N}}\left(\sum_{l=m}^{N-1} a_l z_N^{-kl} + \sum_{l=0}^{m-1} a_l z_N^{-kl}\right) && \textit{since } a_l = a_{l \bmod N} \textit{ and } z_N^{-kl} = z_N^{-k(l \bmod N)} \\
&= \frac{1}{\sqrt{N}}\sum_{l=0}^{N-1} a_l z_N^{-kl} \\
&= A_k && \textit{which also gives the same DFT}
\end{aligned}$$

Remark 2 – Reversed sequence

DFT of reversed sequence is equivalent to *IDFT* of original sequence :

$$\begin{aligned}
seq(B_k) &= F(seq(b_n)) && \textit{suppose } seq(b_n) = rev(seq(a_n)) \\
&= F(rev(seq(a_n))) \\
&= F(rev(\{a_0, a_1, a_2, a_3, \ldots, a_{N-2}, a_{N-1}\})) \\
&= F(\{a_0, a_{-1}, a_{-2}, a_{-3} \ldots, a_{-(N-2)}, a_{-(N-1)}\}) \\
B_k &= \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} a_{-n} z_N^{-kn} \\
&= \frac{1}{\sqrt{N}}\sum_{m=-(N-1)}^{0} a_m z_N^{km} && \textit{put } m = -n, \textit{ reverse summation order} \\
&= \frac{1}{\sqrt{N}}\left(a_0 z_N^{k0} + \sum_{m=-(N-1)}^{-1} a_m z_N^{km}\right) \\
&= \frac{1}{\sqrt{N}}\left(a_0 z_N^{k0} + \sum_{m=1}^{N-1} a_m z_N^{km}\right) && \textit{since } a_m = a_{m+N} \textit{ and } z_N^{km} = z_N^{k(m+N)} \\
&= \frac{1}{\sqrt{N}}\sum_{m=0}^{N-1} a_m z_N^{km} \\
&= F^{-1}(seq(a))
\end{aligned}$$

Summary

A_k	=	$\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} a_n z_N^{-kn}$	=	$\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} a_{n+m} z_N^{-k(n+m)}$	*equation 3*
		$F^{-1}(F(seq(a_n)))$	=	$seq(a_n)$	*equation 4*
		$F(F^{-1}(seq(a_n)))$	=	$seq(a_n)$	*equation 5*
		$F(rev(seq(a_n)))$	=	$F^{-1}(seq(a_n))$	*equation 6*
		$F^{-1}(rev(seq(a_n)))$	=	$F(seq(a_n))$	*equation 7*

Please note that different books may define their DFT and inverse DFT with a different normalisation factor, it is fine as long as the DFT and IDFT match with each other. Here are other possible DFT and IDFT.

(1)	A_k	=	$\frac{1}{N}\sum_{n=0}^{N-1} a_n z_N^{-kn}$	$\forall k \in [0, N-1]$	*DFT*
	a_n	=	$\sum_{k=0}^{N-1} A_k z_N^{kn}$	$\forall n \in [0, N-1]$	*IDFT*
(2)	A_k	=	$\sum_{n=0}^{N-1} a_n z_N^{-kn}$	$\forall k \in [0, N-1]$	*DFT*
	a_n	=	$\frac{1}{N}\sum_{k=0}^{N-1} A_k z_N^{kn}$	$\forall n \in [0, N-1]$	*IDFT*

Property – Circular Convolution

Convolution is applied in (1) filtering and (2) finding probability density function for sum of random variables. Let's define circular convolution of $seq(a_n)$ and $seq(b_n)$, where each sequence has a size of N :

	$seq(c)$	=	$seq(a) \otimes seq(b)$	
if	c_n	=	$\sum_{m=0}^{N-1} a_{n-m} b_m$	*for all* $n \in [0,N-1]$
		=	$\sum_{l=0}^{N-1} a_l b_{n-l}$	*by putting* $l = n-m$

Here is an example for circular convolution. It can be considered as fixing the $seq(b_n)$ and rotating $rev(seq(a_n))$.

Table1	*input*						*output*
$seq(b_n)$	b_0	b_1	b_2	b_3	b_4	b_5	
$seq(a_n)$	a_0	a_5	a_4	a_3	a_2	a_1	$c_0 = a_0b_0 + a_5b_1 + a_4b_2 + a_3b_3 + a_2b_4 + a_1b_5$
	a_1	a_0	a_5	a_4	a_3	a_2	$c_1 = a_1b_0 + a_0b_1 + a_5b_2 + a_4b_3 + a_3b_4 + a_2b_5$
	a_2	a_1	a_0	a_5	a_4	a_3	$c_2 = a_2b_0 + a_1b_1 + a_0b_2 + a_5b_3 + a_4b_4 + a_3b_5$
	a_3	a_2	a_1	a_0	a_5	a_4	$c_3 = a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3 + a_5b_4 + a_4b_5$
	a_4	a_3	a_2	a_1	a_0	a_5	$c_4 = a_4b_0 + a_3b_1 + a_2b_2 + a_1b_3 + a_0b_4 + a_5b_5$
	a_5	a_4	a_3	a_2	a_1	a_0	$c_5 = a_5b_0 + a_4b_1 + a_3b_2 + a_2b_3 + a_1b_4 + a_0b_5$

There is a DFT property which converts convolution into multiplication :

$F(seq(a_n) \otimes seq(b_n))$	=	$\sqrt{N} F(seq(a_n))F(seq(b_n))$	*equation 8*
$F^{-1}(seq(a_n) \otimes seq(b_n))$	=	$\sqrt{N} F^{-1}(seq(a_n))F^{-1}(seq(b_n))$	*equation 9*

Proof : $seq(A_k) = F(seq(a_n))$

$seq(B_k) = F(seq(b_n))$

$seq(C_k) = F(seq(c_n)) = F(seq(a_n) \otimes seq(b_n))$

C_k	=	$\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} c_n z_N^{-kn}$	
	=	$\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}(\sum_{m=0}^{N-1} a_{n-m} b_m) z_N^{-kn}$	
	=	$\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1}(a_{n-m} z_N^{-k(n-m)} b_m z_N^{-km})$	
	=	$\frac{1}{\sqrt{N}}(\sum_{m=0}^{N-1} b_m z_N^{-km}(\sum_{n=0}^{N-1} a_{n-m} z_N^{-k(n-m)}))$	
	=	$(\sum_{m=0}^{N-1} b_m z_N^{-km}) A_k$	*apply equation 3,* A_k *is independent of m*
	=	$\sqrt{N} B_k A_k$	*apply equation 3 again*

Option pricing by DFT

Option pricing using binomial tree (or trinomial tree, or even polynomial tree) can be implemented with DFT. Consider a binomial tree with *N+1* layers (one layer for one timestep, with root vertex in layer *0* as starting time, and *N+1* vertices in layer *N* as maturity, i.e. time $N\Delta t$) parameterised by upward gain u, downward gain d and upward probability p. Lets define the sequence $seq(f_{n,m})$ as the set of all possible underlying price levels in layer n i.e. time $n\Delta t$, where n is the time index and m is underlying price index meaning number of going upward on reaching time $n\Delta t$, hence a smaller m means a lower underlying price.

$$seq(f_{n,m}) = \{ f_{n,m=0}, f_{n,m=1}, ..., f_{n,m=n}, \underbrace{0,0,0,...,0}_{(N+1)-(n+1)} \} \quad where\ n \in [0,N]\ and\ m \in [0,n]$$

where $f_{n,m}$ is defined in equation *10*, which connects derivatives price at time $n\Delta t$ to payoff at maturity, from which equation *11* & *12* are implied, the latter re-formulates derivatives price in form of backward propagation (i.e. dynamic programming) :

$$f_{n,m} = e^{-rn\Delta t}\hat{E}[payoff(s_{N\Delta t}) \mid s_{n\Delta t} = s_0 u^m d^{n-m}] \quad for\ n \in [0,N] \quad equation\ 10$$

$$\Rightarrow\ f_{n,m} = e^{-r\Delta t}(p f_{n+1,m+1} + (1-p) f_{n+1,m}) \quad for\ n \in [0,N-1] \quad equation\ 11$$

$$\Rightarrow\ f_{n,m} = payoff(s_{t+N\Delta t} = s_t u^m d^{N-m}) \quad for\ n = N \quad equation\ 12$$

Backward propagation can be implemented as circular convolution using equation *11* :

$$seq(f_{n,m}) = e^{-r\Delta t} seq(p) \otimes seq(f_{n+1}) \quad for\ n \in [0,N-1] \quad equation\ 13$$

$$seq(p) = \{1-p, \underbrace{0,0,0,...,0}_{N-1}, p\} = rev(\{1-p, p, \underbrace{0,0,0,...,0}_{N-1}\})$$

$$seq(f_{n,m}).size = N+1 \quad for\ n \in [0,N] \quad where\ f_{n,m} = 0, \forall m > n$$

$$seq(p).size = N+1$$

Recall that $seq(f_{n,m})$ has size of *N+1*, we can simply ignore the redundant values $f_{n,m}$ for $m > n$, equation *13* can be verified by checking against *table 1*, putting $b_m = f_{n+1,m}$ and filling $a_0 = 1-p$ and $a_n = p$, while keeping all other elements in $seq(a_n)$ zero, then calculate $seq(c_n) = seq(f_n)$ using circular convolution, repeat the procedure for $n = N-1, N-2, \ldots, 0$. Finally we can read derivative price at time *0* as $f_{0,0}$.

$$seq(f_{0,m}) = e^{-rN\Delta t} seq(f_{N,m}) \otimes \underbrace{seq(p) \otimes ... \otimes seq(p)}_{N} \quad apply\ equation\ 13\ repeatedly$$

$$F^{-1}seq(f_{0,m}) = e^{-rN\Delta t} F^{-1}(seq(f_{N,m}) \otimes \underbrace{seq(p) \otimes ... \otimes seq(p)}_{N}) \quad taking\ inverse\ DFT$$

$$= e^{-rN\Delta t}\sqrt{N+1} F^{-1}[seq(f_{N,m})] F^{-1}[\underbrace{seq(p) \otimes ... \otimes seq(p)}_{N}] \quad apply\ equation\ 9\ repeatedly$$

$$= e^{-rN\Delta t}\sqrt{N+1}^2 F^{-1}[seq(f_{N,m})] F^{-1}[seq(p)] F^{-1}[\underbrace{seq(p) \otimes ... \otimes seq(p)}_{N-1}]$$

$$= e^{-rN\Delta t}\sqrt{N+1}^N F^{-1}[seq(f_{N,m})][F^{-1}[seq(p)]]^N$$

$$\Rightarrow\ seq(f_{0,m}) = e^{-rN\Delta t}(N+1)^{N/2} F[F^{-1}[seq(f_{N,m})][F^{-1}[seq(p)]]^N]$$

Fast Fourier Transform (Cooley-Tukey FFT)

DFTand IDFT can both be naively implemented as matrix multiplication.

$$DFT \quad B = Z_N A \quad where\ A = [a_0\ \ a_1\ \ a_2\ \ ...\ \ a_{N-1}]^T$$

$$IDFT \quad A = Z_N^{-1} B \quad where\ B = [b_0\ \ b_1\ \ b_2\ \ ...\ \ b_{N-1}]^T$$

$$Z_N = \frac{1}{\sqrt{N}} \begin{bmatrix} z_N^{-0\times 0} & z_N^{-0\times 1} & z_N^{-0\times 2} & ... & z_N^{-0\times(N-1)} \\ z_N^{-1\times 0} & z_N^{-1\times 1} & z_N^{-1\times 2} & ... & z_N^{-1\times(N-1)} \\ z_N^{-2\times 0} & z_N^{-2\times 1} & z_N^{-2\times 2} & ... & z_N^{-2\times(N-1)} \\ ... & ... & ... & ... & ... \\ z_N^{-(N-1)\times 0} & z_N^{-(N-1)\times 1} & z_N^{-(N-1)\times 2} & ... & z_N^{-(N-1)\times(N-1)} \end{bmatrix}$$

$$Z_N^{-1} = \frac{1}{\sqrt{N}} \begin{bmatrix} z_N^{0\times 0} & z_N^{0\times 1} & z_N^{0\times 2} & ... & z_N^{0\times(N-1)} \\ z_N^{1\times 0} & z_N^{1\times 1} & z_N^{1\times 2} & ... & z_N^{1\times(N-1)} \\ z_N^{2\times 0} & z_N^{2\times 1} & z_N^{2\times 2} & ... & z_N^{2\times(N-1)} \\ ... & ... & ... & ... & ... \\ z_N^{(N-1)\times 0} & z_N^{(N-1)\times 1} & z_N^{(N-1)\times 2} & ... & z_N^{(N-1)\times(N-1)} \end{bmatrix}$$

You can verify that $Z_N^{-1}Z_N = I$ simply by considering the *ith* row and *jth* column of $Z_N^{-1}Z_N$ *product* :

$$
\begin{aligned}
[Z_N^{-1}Z_N]_{i,j} &= \frac{1}{\sqrt{N}}\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}(z_N^{-i\times n}z_N^{+n\times j}) \\
&= \frac{1}{\sqrt{N}}\frac{1}{\sqrt{N}}\ \underbrace{\sum_{n=0}^{N-1}(z_N^{n(j-i)})}_{\begin{bmatrix} N & if & j-i=const\times N \\ 0 & if & otherwise \end{bmatrix}} && \textit{apply equation 2a and 2b} \\
&= 1_{j-i=const\times N}
\end{aligned}
$$

Since matrix Z_N has a size of $N\times N$ complexity of DFT and IDFT are $O(N^2)$ as both involves N^2 multiplications and $N\times(N-1)$ additions. Fast Fourier Transform is a fast implementation of DFT by taking the advantage when N is a composite number, that is, there exists positive integers N_1 and N_2 such that $N = N_1 \times N_2$. Lets firstly consider DFT of the simplest case : when N is even (note that IDFT can be derived in a similar way).

$$
\begin{aligned}
\textit{Since}\quad z_N^{\pm 2n} &= e^{\pm i(2\pi/N)\times 2n} \\
&= e^{\pm i(2\pi/(N/2))\times n} = z_{N/2}^{\pm n} && \textit{equation 14} \\
A_k &= \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}a_n z_N^{-kn} && \forall k \in [0, N-1] \\
&= \frac{1}{\sqrt{N}}(\overbrace{\sum_{m=0}^{N/2-1}a_{2m}z_N^{-k\times(2m)}}^{even-n}) + \frac{1}{\sqrt{N}}(\overbrace{\sum_{m=0}^{N/2-1}a_{2m+1}z_N^{-k\times(2m+1)}}^{odd-n}) \\
&= \frac{1}{\sqrt{N}}(\sum_{m=0}^{N/2-1}a_{2m}z_N^{-k\times(2m)}) + z_N^{-k}\frac{1}{\sqrt{N}}(\sum_{m=0}^{N/2-1}a_{2m+1}z_N^{-k\times(2m)}) \\
&= \frac{1}{\sqrt{2}}[\frac{1}{\sqrt{N/2}}(\underbrace{\sum_{m=0}^{N/2-1}a_{2m}z_{N/2}^{-km}}_{F[seq(a_{2m})]}) + z_N^{-k}\frac{1}{\sqrt{N/2}}(\underbrace{\sum_{m=0}^{N/2-1}a_{2m+1}z_{N/2}^{-km}}_{F[seq(a_{2m+1})]})] && \textit{apply equation 14} \\
&= \frac{1}{\sqrt{2}}(u_k + z_N^{-k}v_k) && \textit{equation 15}
\end{aligned}
$$

$$
\begin{aligned}
\textit{where}\quad u_k &= \frac{1}{\sqrt{N'}}\sum_{n=0}^{N'-1}a_{2n}z_{N'}^{-kn} && \textit{where } N' = N/2 \\
v_k &= \frac{1}{\sqrt{N'}}\sum_{n=0}^{N'-1}a_{2n+1}z_{N'}^{-kn}
\end{aligned}
$$

The implementation is to partition A into two parts, perform DFT on each part and combine the result using equation *15*. This trick can be recursively applied to matrix U and V (if their sizes are even too) until the size of matrix reaches *2*. In short, we have :

$$
\begin{aligned}
A' &= [a_0 \quad a_2 \quad a_4 \quad \ldots \quad a_{N-2}]^T \\
A'' &= [a_1 \quad a_3 \quad a_5 \quad \ldots \quad a_{N-1}]^T \\
U &= [u_0 \quad u_1 \quad u_2 \quad \ldots \quad u_{N/2-1}]^T && \textit{then } U = Z_{N'}A' \\
V &= [v_0 \quad v_1 \quad v_2 \quad \ldots \quad v_{N/2-1}]^T && \textit{then } V = Z_{N'}A''
\end{aligned}
$$

$$
seq(A_k) = \begin{bmatrix}
(u_0+v_0)/\sqrt{2} \\
(u_1+v_1)/\sqrt{2} \\
\ldots \\
(u_{N'-1}+v_{N'-1})/\sqrt{2} \\
(u_0+z_{N'}^{-1}v_0)/\sqrt{2} \\
(u_1+z_{N'}^{-1}v_1)/\sqrt{2} \\
\ldots \\
(u_{N'-1}+z_{N'}^{-1}v_{N'-1})/\sqrt{2}
\end{bmatrix}
$$

$$
\textit{since}\quad z_{N'}^{-1} = e^{-i\pi} = \cos(-\pi) + i\sin(-\pi) = -1 \qquad \textit{for } N' = 2 \qquad \textit{hence no need to invoke exponential fct}
$$

FFT is thus probably implemented as recursions. When N is the power of *2*, Z_N^{-1} simply equals to *-1*, hence there is no invocation of complex maths function *(no exp, no cos, no sin, just addition and subtraction)*. Let's calculate the complexity *C(N)* recursively, each step is broken down into two subproblems (each has a size of N') together with N additions and N subtractions, hence :

$$
\begin{aligned}
C(N) &= 2C(N/2)+O(N) \\
&= 2(2C(N/2^2)+O(N/2))+O(N) \\
&= 4(2C(N/2^3)+O(N/2^2))+2O(N/2)+O(N) \\
&= O(N\ln N)
\end{aligned}
$$

Recursive implementation is straight forward.

```
template<typename ITER> bool FFT(const ITER& iterA_begin, const ITER& iterA_end, ITER& iterB0)
{
    if (!is_power_of_2(iterA_end – iterA_begin)) return false;
    if (iterA_end – iterA_begin == 2)
    {
        *(iterB0)   = (*iterA_begin + *(iterA_begin+1)) / sqrt2;
        *(iterB0+1) = (*iterA_begin - *(iterA_begin+1)) / sqrt2;
    }
    else
    {
        std::vector<typename std::iterator_traits<ITER>::value_type> A1, U;
        std::vector<typename std::iterator_traits<ITER>::value_type> A2, V;

        partition_into_even_and_odd(iterA_begin, iterA_end, A0, A1);
        if (!FFT(A0.begin(), A0.end(), U)) return false;
        if (!FFT(A1.begin(), A1.end(), V)) return false;

        ITER i0 = A0.begin();
        ITER i1 = A1.begin();
        for(; i0!=A0.end(); ++i0, ++i1, ++iterB0) *iterB0 = (*i0 + *i1) / sqrt2;

        i0 = A0.begin();
        i1 = A1.begin();
        for(; i0!=A0.end(); ++i0, ++i1, ++iterB0) *iterB0 = (*i0 + *i1*inv_zn) / sqrt2;
    }
    return true;
}
```

Iterative implementation is a little bit tricky, as we need to pair up the elements in A.

Reference

[1] *Introduction to Fast Fourier Transform in Finance, A Cerny, Imperial College London.*

[2] *Fast Fourier Transform, Stefan Worner, Swiss Federal Institute of Technology Zurich.*