Christopher Lum
lum@uw.edu

Lecture06f
Fast Fourier Transform

Lecture is on YouTube

The YouTube video entitled ‘Fast Fourier Transform’ that covers this lecture is located at https://youtu.be/yfsSDynscEs

References

Cooley Tukey Algorithm - https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

Introduction

The Discrete Fourier Transform (DFT) is powerful and has an incredibly wide range of applications including but not limited to:

-Signal processing
-Spectrum analysis
-Filtering and convolution
-Image/movie compression
-Correlation analysis
-Audio engineering

Despite this, in the standard form, the DFT computationally expensive. The Fast Fourier Transform (FFT) is an algorithm for computing the DFT in an efficient manner.

The FFT has become such a widely accepted algorithm for computing the DFT that often the two terms are used interchangeably.

Discrete Fourier Transform (DFT) in Matrix Form

Recall the DFT is defined as

$$\hat{f}_k = \sum_{n=0}^{N-1} f_n e^{-i 2\pi k n/N} = \sum_{n=0}^{N-1} f_n \omega_N^{-n k} \qquad k = 0, 1, \ldots, N-1 \qquad \textbf{\textit{(Eq.1.1)}}$$

where $\omega_N = e^{\frac{2\pi i}{N}}$ (primitive Nth root of unit) ***(Eq.1.2)***

```
In[∘]:= ωN[Nn_] = Exp[2 π I / Nn];
```

Note that some texts refer to the primitive N^{th} root of unit as we have in Eq.1.2 (see https://ccrma.stanford.edu/~jos/mdft/Roots_Unity.html and https://en.wikipedia.org/wiki/Root_of_unity) where as other define this as $\omega_N = e^{-\frac{2\pi i}{N}}$ (see https://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf). The notation does not matter as long as we are consistent.

We can write this as

$$\begin{aligned}
\hat{f}_0 &= f_0\, \omega_N^{-0\cdot 0} + f_1\, \omega_N^{-1\cdot 0} + f_2\, \omega_N^{-2\cdot 0} + \ldots + f_{N-1}\, \omega_N^{-(N-1)\cdot 0} \\
\hat{f}_1 &= f_0\, \omega_N^{-0\cdot 1} + f_1\, \omega_N^{-1\cdot 1} + f_2\, \omega_N^{-2\cdot 1} + \ldots + f_{N-1}\, \omega_N^{-(N-1)\cdot 1} \\
\hat{f}_2 &= f_0\, \omega_N^{-0\cdot 2} + f_1\, \omega_N^{-1\cdot 2} + f_2\, \omega_N^{-2\cdot 2} + \ldots + f_{N-1}\, \omega_N^{-(N-1)\cdot 2} \\
&\ldots \\
\hat{f}_{N-1} &= f_0\, \omega_N^{-0\cdot(N-1)} + f_1\, \omega_N^{-1\cdot(N-1)} + f_2\, \omega_N^{-2\cdot(N-1)} + \ldots + f_{N-1}\, \omega_N^{-(N-1)\cdot(N-1)}
\end{aligned}$$

Writing in matrix form yields

$$\begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \ldots \\ \hat{f}_{N-1} \end{pmatrix} = \begin{pmatrix} \omega_N^{-0\cdot 0} & \omega_N^{-1\cdot 0} & \omega_N^{-2\cdot 0} & \ldots & \omega_N^{-(N-1)\cdot 0} \\ \omega_N^{-0\cdot 1} & \omega_N^{-1\cdot 1} & \omega_N^{-2\cdot 1} & \ldots & \omega_N^{-(N-1)\cdot 1} \\ \omega_N^{-0\cdot 2} & \omega_N^{-1\cdot 2} & \omega_N^{-2\cdot 2} & \ldots & \omega_N^{-(N-1)\cdot 2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \omega_N^{-0\cdot(N-1)} & \omega_N^{-1\cdot(N-1)} & \omega_N^{-2\cdot(N-1)} & \ldots & \omega_N^{-(N-1)\cdot(N-1)} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \ldots \\ f_{N-1} \end{pmatrix}$$

$$\begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \ldots \\ \hat{f}_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_N^{-1\cdot 1} & \omega_N^{-2\cdot 1} & \ldots & \omega_N^{-(N-1)\cdot 1} \\ 1 & \omega_N^{-1\cdot 2} & \omega_N^{-2\cdot 2} & \ldots & \omega_N^{-(N-1)\cdot 2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & \omega_N^{-1\cdot(N-1)} & \omega_N^{-2\cdot(N-1)} & \ldots & \omega_N^{-(N-1)\cdot(N-1)} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \ldots \\ f_{N-1} \end{pmatrix}$$

$$\hat{f} = F \bar{f} \qquad \textbf{\textit{(Eq.1.3)}}$$

where $F = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_N^{-1\cdot 1} & \omega_N^{-2\cdot 1} & \ldots & \omega_N^{-(N-1)\cdot 1} \\ 1 & \omega_N^{-1\cdot 2} & \omega_N^{-2\cdot 2} & \ldots & \omega_N^{-(N-1)\cdot 2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & \omega_N^{-1\cdot(N-1)} & \omega_N^{-2\cdot(N-1)} & \ldots & \omega_N^{-(N-1)\cdot(N-1)} \end{pmatrix}$ ***(Eq.1.4)***

$$\bar{f} = \begin{pmatrix} f_0 \\ f_1 \\ \dots \\ f_{N-1} \end{pmatrix} \qquad \hat{f} = \begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \dots \\ \hat{f}_{N-1} \end{pmatrix}$$

Note that the execution of Eq.1.3 is an $O(N^2)$ operation as F is a full, dense matrix.

Here F is a unitary Vandermonde matrix (https://en.wikipedia.org/wiki/Vandermonde_matrix)

```
In[∘]:= DFTMatrix[Nn_] := Module[{F},
    F = IdentityMatrix[Nn];
    For[k = 0, k ≤ Nn - 1, k++,
     For[n = 0, n ≤ Nn - 1, n++,
      F⟦k + 1, n + 1⟧ = ωN[Nn]^(-n k);
     ]
    ];

    (*Output the matrix*)
    F
   ];
```

It will also be useful later to define a matrix to create a matrix of all zeros

```
In[∘]:= ZeroMatrix[Nn_] := Module[{A},
    A = IdentityMatrix[Nn];
    For[k = 0, k ≤ Nn - 1, k++,
     For[n = 0, n ≤ Nn - 1, n++,
      A⟦k + 1, n + 1⟧ = 0;
     ]
    ];

    (*Output the matrix*)
    A
   ];
```

Fast Fourier Transform (FFT)

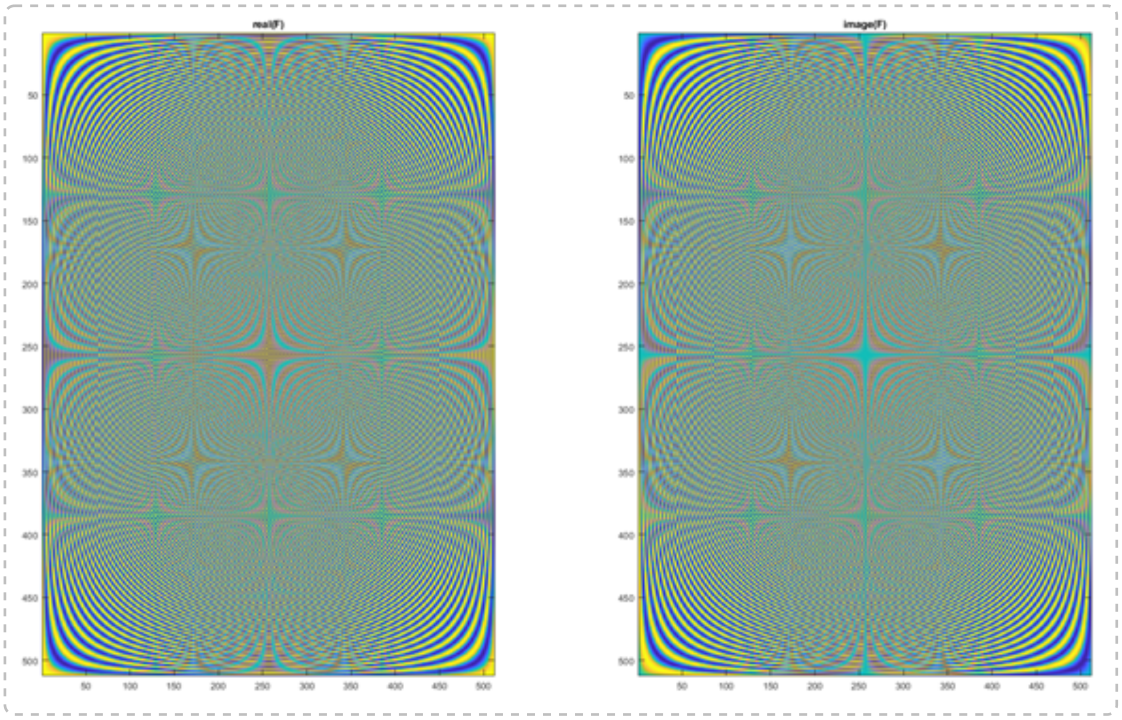
Note that the DFT matrix can be computed based on only the number of samples. The actual sample values are not required in Eq.1.4

In[∘]:=
```
NGiven = 8;
FGiven = DFTMatrix[NGiven];
FGiven // MatrixForm
```

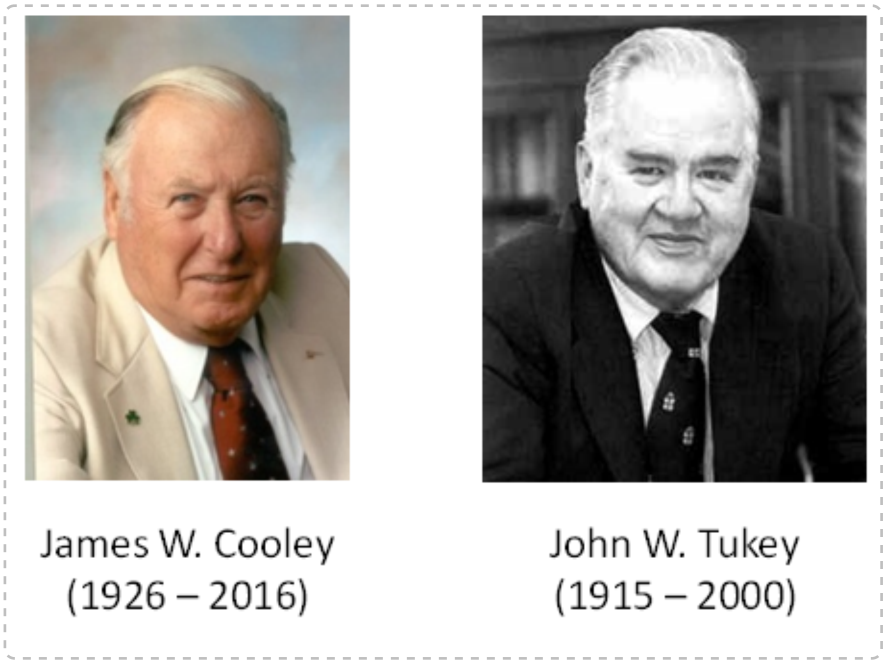
Out[∘]//MatrixForm=

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & e^{-\frac{i\pi}{4}} & -i & e^{-\frac{3i\pi}{4}} & -1 & e^{\frac{3i\pi}{4}} & i & e^{\frac{i\pi}{4}} \\ 1 & -i & -1 & i & 1 & -i & -1 & i \\ 1 & e^{-\frac{3i\pi}{4}} & i & e^{-\frac{i\pi}{4}} & -1 & e^{\frac{i\pi}{4}} & -i & e^{\frac{3i\pi}{4}} \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & e^{\frac{3i\pi}{4}} & -i & e^{\frac{i\pi}{4}} & -1 & e^{-\frac{i\pi}{4}} & i & e^{-\frac{3i\pi}{4}} \\ 1 & i & -1 & -i & 1 & i & -1 & -i \\ 1 & e^{\frac{i\pi}{4}} & i & e^{\frac{3i\pi}{4}} & -1 & e^{-\frac{3i\pi}{4}} & -i & e^{-\frac{i\pi}{4}} \end{pmatrix}$$

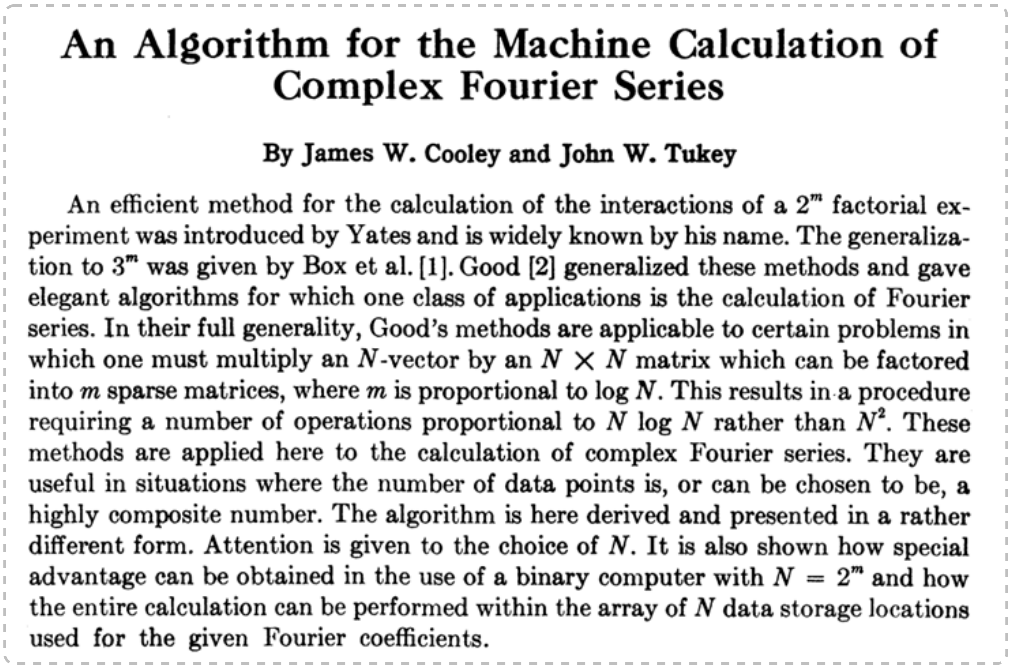
We can visualize this matrix by examining the real and imaginary parts of this matrix (shown below for $N = 512$)



Examining these plots we see that there are patterns within this matrix structure. Recall from our previous discussion on N^{th} roots of unity showed that there were significant repeats in the entries of the DFT matrix. Cooley and Tukey exploited this in their famous FFT algorithm (https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm)



Cooley, James W. and Tukey, John W., “An Algorithm for the Machine Calculation of Complex Fourier Series,” Mathematics of Computation, Vol. 19, April 1965, pages 297-301.



An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2. These methods are applied here to the calculation of complex Fourier series. They are useful in situations where the number of data points is, or can be chosen to be, a highly composite number. The algorithm is here derived and presented in a rather different form. Attention is given to the choice of N. It is also shown how special advantage can be obtained in the use of a binary computer with $N = 2^m$ and how the entire calculation can be performed within the array of N data storage locations used for the given Fourier coefficients.

Returning to our $N = 8$ case, recall that we can write $\overline{f}$ and $\hat{f}$ as vectors

$$\bar{f} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{pmatrix} \qquad \hat{f} = \begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \hat{f}_4 \\ \hat{f}_5 \\ \hat{f}_6 \\ \hat{f}_7 \end{pmatrix}$$

In[ ]:= $$\mathtt{f} = \begin{pmatrix} \mathtt{f0} \\ \mathtt{f1} \\ \mathtt{f2} \\ \mathtt{f3} \\ \mathtt{f4} \\ \mathtt{f5} \\ \mathtt{f6} \\ \mathtt{f7} \end{pmatrix}; \; \mathtt{fhat} = \begin{pmatrix} \mathtt{fhat0} \\ \mathtt{fhat1} \\ \mathtt{fhat2} \\ \mathtt{fhat3} \\ \mathtt{fhat4} \\ \mathtt{fhat5} \\ \mathtt{fhat6} \\ \mathtt{fhat7} \end{pmatrix};$$

Let us order this so the even indices ($n = 2\,m$) are stacked on top of the odd indices ($n = 2\,m + 1$)

$$\bar{f}_R = \begin{pmatrix} \bar{f}_{\text{even}} \\ \bar{f}_{\text{odd}} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_2 \\ f_4 \\ f_6 \\ f_1 \\ f_3 \\ f_5 \\ f_7 \end{pmatrix} \qquad R = \text{re} - \text{ordered}$$

We see that we can achieve this using the following transformation matrix

$$\begin{pmatrix} f_0 \\ f_2 \\ f_4 \\ f_6 \\ f_1 \\ f_3 \\ f_5 \\ f_7 \end{pmatrix} = \begin{pmatrix} 1 & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & 1 & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & 1 & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & 1 & \square \\ \square & 1 & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & 1 & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & 1 & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & 1 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{pmatrix}$$

$$\bar{f}_R = T\,\bar{f} \qquad \textbf{\textit{(Eq.1.5)}}$$

```
In[•]:= TMatrix[Nn_] := Module[{T},
          T = ZeroMatrix[Nn];

          For[k = 0, k ≤ Nn - 1, k++,
           row = k + 1;
           If[row ≤ Nn / 2,
             (*even indices (top half of matrix*)
            idx = row * 2 - 1;
            ,
             (*odd indices (bottom half of matrix*)
            idx = (row - Nn / 2) * 2;
           ];
           T⟦row, idx⟧ = 1;
          ];

          (*Output matrix*)
          T
         ];
```

```
In[•]:= T = TMatrix[NGiven];
        T // MatrixForm
```

Out[•]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Recall from our discussion on elementary row operations (see video entitled ‘Elementary Row Operations, Row Echelon Form, and Reduced Row Echelon Form’ at https://youtu.be/AxgzzJposVo) that the elementary operation of switching columns can be implemented by right multiplying the matrix in question with T^T

```
In[∘]:= (*Create a matrix where we can easily see the columns*)
        A = ZeroMatrix[NGiven];
        For[k = 0, k ≤ NGiven - 1, k++,
         For[n = 0, n ≤ NGiven - 1, n++,
          A⟦k + 1, n + 1⟧ = (n + 1) + (k / 100);
         ]
        ]

        Print["Original Matrix"]
        A // MatrixForm // N

        (*Perform switching of columns by right multiplying with T*)
        AR = A.Transpose[T];

        Print["Matrix with reordered columns"];
        AR // MatrixForm // N
```

```
        Original Matrix
Out[∘]//MatrixForm=
        ( 1.    2.    3.    4.    5.    6.    7.    8.  )
        ( 1.01  2.01  3.01  4.01  5.01  6.01  7.01  8.01 )
        ( 1.02  2.02  3.02  4.02  5.02  6.02  7.02  8.02 )
        ( 1.03  2.03  3.03  4.03  5.03  6.03  7.03  8.03 )
        ( 1.04  2.04  3.04  4.04  5.04  6.04  7.04  8.04 )
        ( 1.05  2.05  3.05  4.05  5.05  6.05  7.05  8.05 )
        ( 1.06  2.06  3.06  4.06  5.06  6.06  7.06  8.06 )
        ( 1.07  2.07  3.07  4.07  5.07  6.07  7.07  8.07 )

        Matrix with reordered columns
Out[∘]//MatrixForm=
        ( 1.    3.    5.    7.    2.    4.    6.    8.  )
        ( 1.01  3.01  5.01  7.01  2.01  4.01  6.01  8.01 )
        ( 1.02  3.02  5.02  7.02  2.02  4.02  6.02  8.02 )
        ( 1.03  3.03  5.03  7.03  2.03  4.03  6.03  8.03 )
        ( 1.04  3.04  5.04  7.04  2.04  4.04  6.04  8.04 )
        ( 1.05  3.05  5.05  7.05  2.05  4.05  6.05  8.05 )
        ( 1.06  3.06  5.06  7.06  2.06  4.06  6.06  8.06 )
        ( 1.07  3.07  5.07  7.07  2.07  4.07  6.07  8.07 )
```

We can therefore reorder the columns of F using this transformation

$$F_R = F\,T^T \qquad \textbf{\textit{(Eq.1.6)}}$$

In[∘]:=
```
FRGiven = FGiven.Transpose[T];
FRGiven // MatrixForm
```

Out[∘]//MatrixForm=

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i & e^{-\frac{i\pi}{4}} & e^{-\frac{3i\pi}{4}} & e^{\frac{3i\pi}{4}} & e^{\frac{i\pi}{4}} \\ 1 & -1 & 1 & -1 & -i & i & -i & i \\ 1 & i & -1 & -i & e^{-\frac{3i\pi}{4}} & e^{-\frac{i\pi}{4}} & e^{\frac{i\pi}{4}} & e^{\frac{3i\pi}{4}} \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -i & -1 & i & e^{\frac{3i\pi}{4}} & e^{\frac{i\pi}{4}} & e^{-\frac{i\pi}{4}} & e^{-\frac{3i\pi}{4}} \\ 1 & -1 & 1 & -1 & i & -i & i & -i \\ 1 & i & -1 & -i & e^{\frac{i\pi}{4}} & e^{\frac{3i\pi}{4}} & e^{-\frac{3i\pi}{4}} & e^{-\frac{i\pi}{4}} \end{pmatrix}$$

We can also reorder the vector $\overline{f}$

In[∘]:=
```
fR = T.f;
fR // MatrixForm
```

Out[∘]//MatrixForm=

$$\begin{pmatrix} f0 \\ f2 \\ f4 \\ f6 \\ f1 \\ f3 \\ f5 \\ f7 \end{pmatrix}$$

So we have

$$\hat{f} = F_R \overline{f}_R \qquad \textbf{(Eq.1.7)}$$

In[∘]:=
```
FRGiven.fR == FGiven.f
```

Out[∘]=
```
True
```

We can see that we recover the original equation if we insert the definitions of F_R and $\overline{f}_R$

$$\hat{f} = \left(F\, T^T\right)\left(T\, \overline{f}\right) \qquad \text{recall: } T \text{ is unitary so } T^T\, T = I$$

$$\hat{f} = F\, \overline{f}$$

We now make the observation that F_R can be written as

$$F_R = F_{R,l}\, R_{R,r}$$

$$F_R = \begin{pmatrix} I_{\text{HalfN}} & D_{\text{HalfN}} \\ I_{\text{HalfN}} & -D_{\text{HalfN}} \end{pmatrix} \begin{pmatrix} F_{\text{HalfN}} & 0_{\text{HalfN}} \\ 0_{\text{HalfN}} & F_{\text{HalfN}} \end{pmatrix} \qquad \textbf{(Eq.1.8)}$$

where $D_{\text{HalfN}} = \begin{pmatrix} \omega_N{}^{-0} & 0 & 0 & 0 \\ 0 & \omega_N{}^{-1} & 0 & 0 \\ 0 & 0 & \omega_N{}^{-2} & 0 \\ 0 & 0 & 0 & \omega_N{}^{-3} \end{pmatrix}$

$$F_{\text{HalfN}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_{N/2}{}^{-1\cdot 1} & \omega_{N/2}{}^{-2\cdot 1} & \omega_{N/2}{}^{-3\cdot 1} \\ 1 & \omega_{N/2}{}^{-1\cdot 2} & \omega_{N/2}{}^{-2\cdot 2} & \omega_{N/2}{}^{-3\cdot 2} \\ 1 & \omega_{N/2}{}^{-1\cdot 3} & \omega_{N/2}{}^{-2\cdot 3} & \omega_{N/2}{}^{-3\cdot 3} \end{pmatrix} = \text{standard 4 x4 DFT matrix}$$

Note that D_{HalfN} uses ω_N whereas F_{HalfN} uses $\omega_{N/2}$

In[∘]:=
```
(*0HalfN*)
ZeroHalfNGiven = ZeroMatrix[NGiven / 2];

Print["0HalfN"]
ZeroHalfNGiven // MatrixForm

(*IHalfN*)
IHalfNGiven = IdentityMatrix[NGiven / 2];

Print["IHalfN"]
IHalfNGiven // MatrixForm

(*DHalfN*)
ωNHalfNListGiven = ConstantArray[0, NGiven / 2];
For[k = 0, k ≤ NGiven / 2 - 1, k++,
  ωNHalfNListGiven[[k + 1]] = ωN[NGiven]^-k;
]
DHalfNGiven = DiagonalMatrix[ωNHalfNListGiven];

Print["DHalfN"]
DHalfNGiven // MatrixForm

(*FHalfN*)
FHalfNGiven = DFTMatrix[NGiven / 2];

Print["FHalfN"]
FHalfNGiven // MatrixForm
```

0_{HalfN}

Out[∘]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

I_{HalfN}

Out[∘]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

D_{HalfN}

Out[∘]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-\frac{i\pi}{4}} & 0 & 0 \\ 0 & 0 & -i & 0 \\ 0 & 0 & 0 & e^{-\frac{3i\pi}{4}} \end{pmatrix}$$

F_{HalfN}

Out[∘]//MatrixForm=

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

Verify that this works

In[∘]:=
```
(*Compute F_{R,l}*)
tempLeft = Join[IHalfNGiven, IHalfNGiven];
tempRight = Join[DHalfNGiven, -DHalfNGiven];

FRlGiven = Join[Transpose[tempLeft], Transpose[tempRight]] // Transpose;

(*Compute F_{R,r}*)
tempLeft = Join[FHalfNGiven, ZeroHalfNGiven];
tempRight = Join[ZeroHalfNGiven, FHalfNGiven];

FRrGiven = Join[Transpose[tempLeft], Transpose[tempRight]] // Transpose;;

(*Check calculation*)
FRGivenCheck = FRlGiven.FRrGiven;

Print["Difference"]
FRGiven - FRGivenCheck // Simplify // MatrixForm
```

```
Difference
```

Out[∘]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Let us use Eq.1.8 to rewrite the computation of the DFT as

$$\hat{f} = F_R \bar{f}_R$$

$$\hat{f} = \begin{pmatrix} I_{\text{HalfN}} & D_{\text{HalfN}} \\ I_{\text{HalfN}} & -D_{\text{HalfN}} \end{pmatrix} \begin{pmatrix} F_{\text{HalfN}} & 0_{\text{HalfN}} \\ 0_{\text{HalfN}} & F_{\text{HalfN}} \end{pmatrix} \begin{pmatrix} \bar{f}_{\text{even}} \\ \bar{f}_{\text{odd}} \end{pmatrix}$$

The benefit of this formulation is that instead of a full $O\left(N^2\right)$ operation, the difficult part of this operation is only $2\,O\left(\left(\frac{N}{2}\right)^2\right)$ operations

$F_{\text{HalfN}}\,\bar{f}_{\text{even}}$ ***(Eq.1.9)***

$F_{\text{HalfN}}\,\bar{f}_{\text{odd}}$ ***(Eq.1.10)***

Eq.1.9 is a smaller DFT calculation which both uses F_{HalfN}. This matrix can be split again by half using the same procedure as above.

This also works for larger matrices. For example, let us increase $N = 16$

```
In[∘]:= NGiven16 = 16;
FGiven16 = DFTMatrix[NGiven16];
Print["F16"]
FGiven16 // MatrixForm

T16 = TMatrix[NGiven16];
Print["T16"]
T16 // MatrixForm

FRGiven16 = FGiven16.Transpose[T16];
Print["FR16"]
FRGiven16 // MatrixForm
```

```
F16
```

Out[∘]//MatrixForm=

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & e^{-\frac{i\pi}{8}} & e^{-\frac{i\pi}{4}} & e^{-\frac{3i\pi}{8}} & -i & e^{-\frac{5i\pi}{8}} & e^{-\frac{3i\pi}{4}} & e^{-\frac{7i\pi}{8}} & -1 & e^{\frac{7i\pi}{8}} & e^{\frac{3i\pi}{4}} & e^{\frac{5i\pi}{8}} & i & e^{\frac{3i\pi}{8}} & e^{\frac{i\pi}{4}} & e^{\frac{i\pi}{8}} \\
1 & e^{-\frac{i\pi}{4}} & -i & e^{-\frac{3i\pi}{4}} & -1 & e^{\frac{3i\pi}{4}} & i & e^{\frac{i\pi}{4}} & 1 & e^{-\frac{i\pi}{4}} & -i & e^{-\frac{3i\pi}{4}} & -1 & e^{\frac{3i\pi}{4}} & i & e^{\frac{i\pi}{4}} \\
1 & e^{-\frac{3i\pi}{8}} & e^{-\frac{3i\pi}{4}} & e^{\frac{7i\pi}{8}} & i & e^{\frac{i\pi}{8}} & e^{-\frac{i\pi}{4}} & e^{-\frac{5i\pi}{8}} & -1 & e^{\frac{5i\pi}{8}} & e^{\frac{i\pi}{4}} & e^{-\frac{i\pi}{8}} & -i & e^{-\frac{7i\pi}{8}} & e^{\frac{3i\pi}{4}} & e^{\frac{3i\pi}{8}} \\
1 & -i & -1 & i & 1 & -i & -1 & i & 1 & -i & -1 & i & 1 & -i & -1 & i \\
1 & e^{-\frac{5i\pi}{8}} & e^{\frac{3i\pi}{4}} & e^{\frac{i\pi}{8}} & -i & e^{\frac{7i\pi}{8}} & e^{\frac{i\pi}{4}} & e^{-\frac{3i\pi}{8}} & -1 & e^{\frac{3i\pi}{8}} & e^{-\frac{i\pi}{4}} & e^{-\frac{7i\pi}{8}} & i & e^{-\frac{i\pi}{8}} & e^{-\frac{3i\pi}{4}} & e^{\frac{5i\pi}{8}} \\
1 & e^{-\frac{3i\pi}{4}} & i & e^{-\frac{i\pi}{4}} & -1 & e^{\frac{i\pi}{4}} & -i & e^{\frac{3i\pi}{4}} & 1 & e^{-\frac{3i\pi}{4}} & i & e^{-\frac{i\pi}{4}} & -1 & e^{\frac{i\pi}{4}} & -i & e^{\frac{3i\pi}{4}} \\
1 & e^{-\frac{7i\pi}{8}} & e^{\frac{i\pi}{4}} & e^{-\frac{5i\pi}{8}} & i & e^{-\frac{3i\pi}{8}} & e^{\frac{3i\pi}{4}} & e^{-\frac{i\pi}{8}} & -1 & e^{\frac{i\pi}{8}} & e^{-\frac{3i\pi}{4}} & e^{\frac{3i\pi}{8}} & -i & e^{\frac{5i\pi}{8}} & e^{-\frac{i\pi}{4}} & e^{\frac{7i\pi}{8}} \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & e^{\frac{7i\pi}{8}} & e^{-\frac{i\pi}{4}} & e^{\frac{5i\pi}{8}} & -i & e^{\frac{3i\pi}{8}} & e^{-\frac{3i\pi}{4}} & e^{\frac{i\pi}{8}} & -1 & e^{-\frac{i\pi}{8}} & e^{\frac{3i\pi}{4}} & e^{-\frac{3i\pi}{8}} & i & e^{-\frac{5i\pi}{8}} & e^{\frac{i\pi}{4}} & e^{-\frac{7i\pi}{8}} \\
1 & e^{\frac{3i\pi}{4}} & -i & e^{\frac{i\pi}{4}} & -1 & e^{-\frac{i\pi}{4}} & i & e^{-\frac{3i\pi}{4}} & 1 & e^{\frac{3i\pi}{4}} & -i & e^{\frac{i\pi}{4}} & -1 & e^{-\frac{i\pi}{4}} & i & e^{-\frac{3i\pi}{4}} \\
1 & e^{\frac{5i\pi}{8}} & e^{-\frac{3i\pi}{4}} & e^{-\frac{i\pi}{8}} & i & e^{-\frac{7i\pi}{8}} & e^{-\frac{i\pi}{4}} & e^{\frac{3i\pi}{8}} & -1 & e^{-\frac{3i\pi}{8}} & e^{\frac{i\pi}{4}} & e^{\frac{7i\pi}{8}} & -i & e^{\frac{i\pi}{8}} & e^{\frac{3i\pi}{4}} & e^{-\frac{5i\pi}{8}} \\
1 & i & -1 & -i & 1 & i & -1 & -i & 1 & i & -1 & -i & 1 & i & -1 & -i \\
1 & e^{\frac{3i\pi}{8}} & e^{\frac{3i\pi}{4}} & e^{-\frac{7i\pi}{8}} & -i & e^{-\frac{i\pi}{8}} & e^{\frac{i\pi}{4}} & e^{\frac{5i\pi}{8}} & -1 & e^{-\frac{5i\pi}{8}} & e^{-\frac{i\pi}{4}} & e^{\frac{i\pi}{8}} & i & e^{\frac{7i\pi}{8}} & e^{-\frac{3i\pi}{4}} & e^{-\frac{3i\pi}{8}} \\
1 & e^{\frac{i\pi}{4}} & i & e^{\frac{3i\pi}{4}} & -1 & e^{-\frac{3i\pi}{4}} & -i & e^{-\frac{i\pi}{4}} & 1 & e^{\frac{i\pi}{4}} & i & e^{\frac{3i\pi}{4}} & -1 & e^{-\frac{3i\pi}{4}} & -i & e^{-\frac{i\pi}{4}} \\
1 & e^{\frac{i\pi}{8}} & e^{\frac{i\pi}{4}} & e^{\frac{3i\pi}{8}} & i & e^{\frac{5i\pi}{8}} & e^{\frac{3i\pi}{4}} & e^{\frac{7i\pi}{8}} & -1 & e^{-\frac{7i\pi}{8}} & e^{-\frac{3i\pi}{4}} & e^{-\frac{5i\pi}{8}} & -i & e^{-\frac{3i\pi}{8}} & e^{-\frac{i\pi}{4}} & e^{-\frac{i\pi}{8}}
\end{pmatrix}$$

T_{16}

Out[∘]//MatrixForm=

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

F_{R16}

Out[∘]//MatrixForm=

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & e^{-\frac{i\pi}{4}} & -i & e^{-\frac{3i\pi}{4}} & -1 & e^{\frac{3i\pi}{4}} & i & e^{\frac{i\pi}{4}} & e^{-\frac{i\pi}{8}} & e^{-\frac{3i\pi}{8}} & e^{-\frac{5i\pi}{8}} & e^{-\frac{7i\pi}{8}} & e^{\frac{7i\pi}{8}} & e^{\frac{5i\pi}{8}} & e^{\frac{3i\pi}{8}} & e^{\frac{i\pi}{8}} \\
1 & -i & -1 & i & 1 & -i & -1 & i & e^{-\frac{i\pi}{4}} & e^{-\frac{3i\pi}{4}} & e^{\frac{3i\pi}{4}} & e^{\frac{i\pi}{4}} & e^{-\frac{i\pi}{4}} & e^{-\frac{3i\pi}{4}} & e^{\frac{3i\pi}{4}} & e^{\frac{i\pi}{4}} \\
1 & e^{-\frac{3i\pi}{4}} & i & e^{-\frac{i\pi}{4}} & -1 & e^{\frac{i\pi}{4}} & -i & e^{\frac{3i\pi}{4}} & e^{-\frac{3i\pi}{8}} & e^{\frac{7i\pi}{8}} & e^{\frac{i\pi}{8}} & e^{-\frac{5i\pi}{8}} & e^{\frac{5i\pi}{8}} & e^{-\frac{i\pi}{8}} & e^{-\frac{7i\pi}{8}} & e^{\frac{3i\pi}{8}} \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -i & i & -i & i & -i & i & -i & i \\
1 & e^{\frac{3i\pi}{4}} & -i & e^{\frac{i\pi}{4}} & -1 & e^{-\frac{i\pi}{4}} & i & e^{-\frac{3i\pi}{4}} & e^{-\frac{5i\pi}{8}} & e^{\frac{i\pi}{8}} & e^{\frac{7i\pi}{8}} & e^{-\frac{3i\pi}{8}} & e^{\frac{3i\pi}{8}} & e^{-\frac{7i\pi}{8}} & e^{-\frac{i\pi}{8}} & e^{\frac{5i\pi}{8}} \\
1 & i & -1 & -i & 1 & i & -1 & -i & e^{-\frac{3i\pi}{4}} & e^{-\frac{i\pi}{4}} & e^{\frac{i\pi}{4}} & e^{\frac{3i\pi}{4}} & e^{-\frac{3i\pi}{4}} & e^{-\frac{i\pi}{4}} & e^{\frac{i\pi}{4}} & e^{\frac{3i\pi}{4}} \\
1 & e^{\frac{i\pi}{4}} & i & e^{\frac{3i\pi}{4}} & -1 & e^{-\frac{3i\pi}{4}} & -i & e^{-\frac{i\pi}{4}} & e^{-\frac{7i\pi}{8}} & e^{-\frac{5i\pi}{8}} & e^{-\frac{3i\pi}{8}} & e^{-\frac{i\pi}{8}} & e^{\frac{i\pi}{8}} & e^{\frac{3i\pi}{8}} & e^{\frac{5i\pi}{8}} & e^{\frac{7i\pi}{8}} \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1 & e^{-\frac{i\pi}{4}} & -i & e^{-\frac{3i\pi}{4}} & -1 & e^{\frac{3i\pi}{4}} & i & e^{\frac{i\pi}{4}} & e^{\frac{7i\pi}{8}} & e^{\frac{5i\pi}{8}} & e^{\frac{3i\pi}{8}} & e^{\frac{i\pi}{8}} & e^{-\frac{i\pi}{8}} & e^{-\frac{3i\pi}{8}} & e^{-\frac{5i\pi}{8}} & e^{-\frac{7i\pi}{8}} \\
1 & -i & -1 & i & 1 & -i & -1 & i & e^{\frac{3i\pi}{4}} & e^{\frac{i\pi}{4}} & e^{-\frac{i\pi}{4}} & e^{-\frac{3i\pi}{4}} & e^{\frac{3i\pi}{4}} & e^{\frac{i\pi}{4}} & e^{-\frac{i\pi}{4}} & e^{-\frac{3i\pi}{4}} \\
1 & e^{-\frac{3i\pi}{4}} & i & e^{-\frac{i\pi}{4}} & -1 & e^{\frac{i\pi}{4}} & -i & e^{\frac{3i\pi}{4}} & e^{\frac{5i\pi}{8}} & e^{-\frac{i\pi}{8}} & e^{-\frac{7i\pi}{8}} & e^{\frac{3i\pi}{8}} & e^{-\frac{3i\pi}{8}} & e^{\frac{7i\pi}{8}} & e^{\frac{i\pi}{8}} & e^{-\frac{5i\pi}{8}} \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & i & -i & i & -i & i & -i & i & -i \\
1 & e^{\frac{3i\pi}{4}} & -i & e^{\frac{i\pi}{4}} & -1 & e^{-\frac{i\pi}{4}} & i & e^{-\frac{3i\pi}{4}} & e^{\frac{3i\pi}{8}} & e^{-\frac{7i\pi}{8}} & e^{-\frac{i\pi}{8}} & e^{\frac{5i\pi}{8}} & e^{-\frac{5i\pi}{8}} & e^{\frac{i\pi}{8}} & e^{\frac{7i\pi}{8}} & e^{-\frac{3i\pi}{8}} \\
1 & i & -1 & -i & 1 & i & -1 & -i & e^{\frac{i\pi}{4}} & e^{\frac{3i\pi}{4}} & e^{-\frac{3i\pi}{4}} & e^{-\frac{i\pi}{4}} & e^{\frac{i\pi}{4}} & e^{\frac{3i\pi}{4}} & e^{-\frac{3i\pi}{4}} & e^{-\frac{i\pi}{4}} \\
1 & e^{\frac{i\pi}{4}} & i & e^{\frac{3i\pi}{4}} & -1 & e^{-\frac{3i\pi}{4}} & -i & e^{-\frac{i\pi}{4}} & e^{\frac{i\pi}{8}} & e^{\frac{3i\pi}{8}} & e^{\frac{5i\pi}{8}} & e^{\frac{7i\pi}{8}} & e^{-\frac{7i\pi}{8}} & e^{-\frac{5i\pi}{8}} & e^{-\frac{3i\pi}{8}} & e^{-\frac{i\pi}{8}}
\end{pmatrix}$$

In[∘]:=
```
(*0HalfN*)
ZeroHalfNGiven16 = ZeroMatrix[NGiven16 / 2];

Print["0HalfN16"]
ZeroHalfNGiven16 // MatrixForm

(*IHalfN*)
IHalfNGiven16 = IdentityMatrix[NGiven16 / 2];

Print["IHalfN16"]
IHalfNGiven16 // MatrixForm

(*DHalfN*)
ωNHalfNListGiven16 = ConstantArray[0, NGiven16 / 2];
For[k = 0, k ≤ NGiven16 / 2 - 1, k++,
 ωNHalfNListGiven16[[k + 1]] = ωN[NGiven16]^-k;
]
DHalfNGiven16 = DiagonalMatrix[ωNHalfNListGiven16];

Print["DHalfN16"]
DHalfNGiven16 // MatrixForm

(*FHalfN*)
FHalfNGiven16 = DFTMatrix[NGiven16 / 2];

Print["FHalfN16"]
FHalfNGiven16 // MatrixForm
```

0_{HalfN16}

Out[∘]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

I_{HalfN16}

Out[ ]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

D_{HalfN16}

Out[ ]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{-\frac{i\pi}{8}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & e^{-\frac{i\pi}{4}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{-\frac{3i\pi}{8}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{-\frac{5i\pi}{8}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & e^{-\frac{3i\pi}{4}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{-\frac{7i\pi}{8}} \end{pmatrix}$$

F_{HalfN16}

Out[ ]//MatrixForm=

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & e^{-\frac{i\pi}{4}} & -i & e^{-\frac{3i\pi}{4}} & -1 & e^{\frac{3i\pi}{4}} & i & e^{\frac{i\pi}{4}} \\ 1 & -i & -1 & i & 1 & -i & -1 & i \\ 1 & e^{-\frac{3i\pi}{4}} & i & e^{-\frac{i\pi}{4}} & -1 & e^{\frac{i\pi}{4}} & -i & e^{\frac{3i\pi}{4}} \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & e^{\frac{3i\pi}{4}} & -i & e^{\frac{i\pi}{4}} & -1 & e^{-\frac{i\pi}{4}} & i & e^{-\frac{3i\pi}{4}} \\ 1 & i & -1 & -i & 1 & i & -1 & -i \\ 1 & e^{\frac{i\pi}{4}} & i & e^{\frac{3i\pi}{4}} & -1 & e^{-\frac{3i\pi}{4}} & -i & e^{-\frac{i\pi}{4}} \end{pmatrix}$$

Verify that this works

```
In[∘]:= (*Compute F_{R,l}*)
        tempLeft16 = Join[IHalfNGiven16, IHalfNGiven16];
        tempRight16 = Join[DHalfNGiven16, -DHalfNGiven16];

        FRlGiven16 = Join[Transpose[tempLeft16], Transpose[tempRight16]] // Transpose;

        (*Compute F_{R,r}*)
        tempLeft16 = Join[FHalfNGiven16, ZeroHalfNGiven16];
        tempRight16 = Join[ZeroHalfNGiven16, FHalfNGiven16];

        FRrGiven16 = Join[Transpose[tempLeft16], Transpose[tempRight16]] // Transpose;;

        (*Check calculation*)
        FRGivenCheck16 = FRlGiven16.FRrGiven16;

        Print["Difference"]
        FRGiven16 - FRGivenCheck16 // Simplify // MatrixForm

        Difference
```

Out[∘]//MatrixForm=

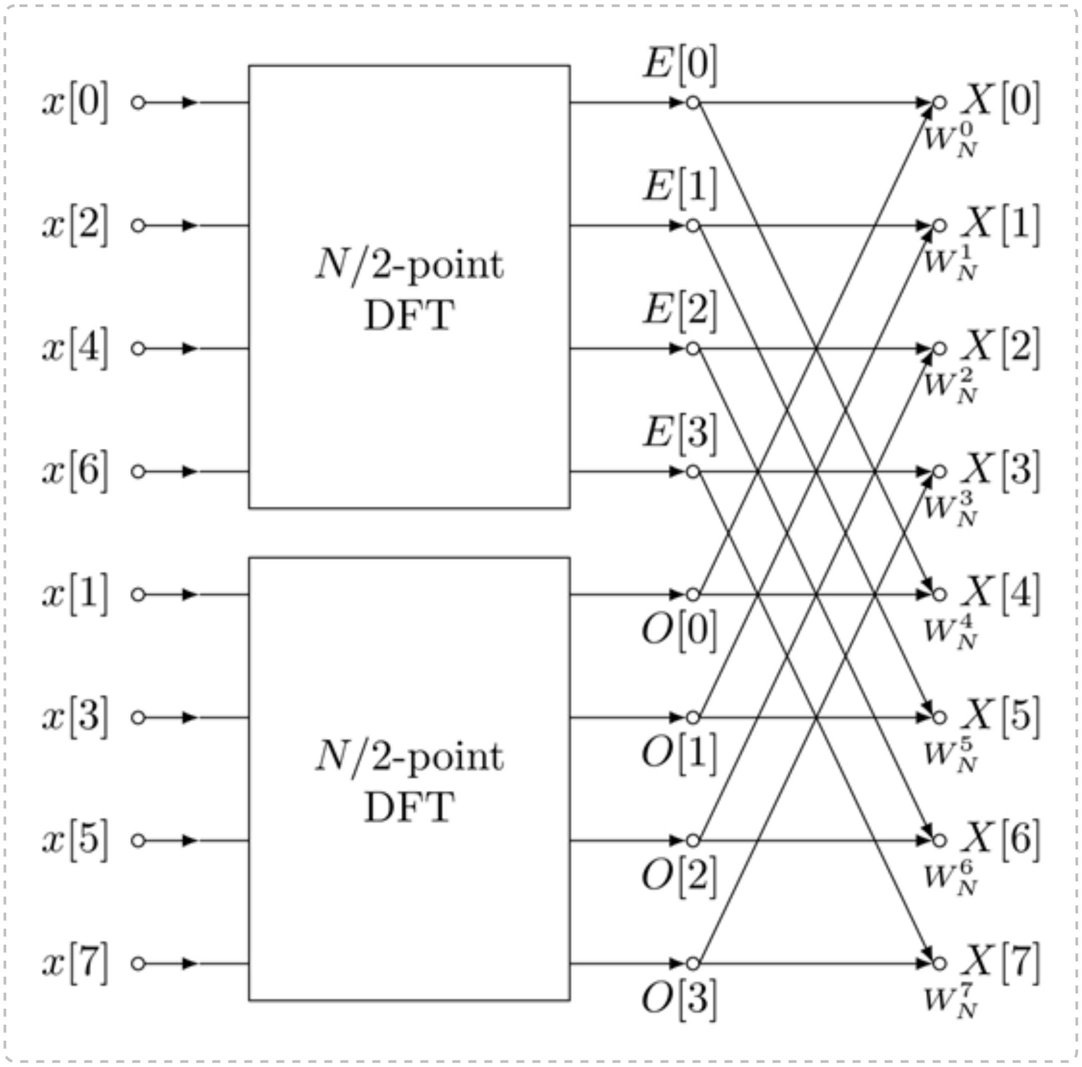
$$\begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix}$$

The benefit is that the F_{HalfN} term in the $N = 16$ case is the same as the F matrix we examined earlier with $N = 8$ which we demonstrated can be split in two.

```
In[∘]:= FHalfNGiven16 == FGiven // Simplify
Out[∘]= True
```

We can repeat this process for powers of 2.

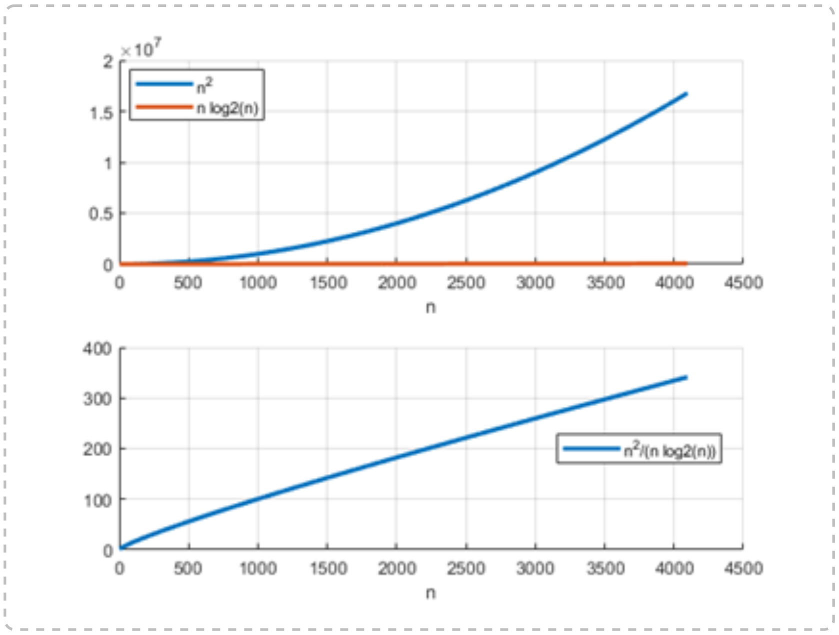
This leads to the FFT algorithm which basically halves the DFT problem successively to speed up computations. This process is often visualized as the FFT Butterfly diagram as shown below.



As such, the FFT approach is $O(N \log N)$ instead of $O(N^2)$

FFT	$\Longleftrightarrow$	$O(N \log N)$	(base 2)
DFT matrix	$\Longleftrightarrow$	$O(N^2)$	

The savings is shown below



For example, if $N = 2^{15}$ (32,768). Considering that most audio is sampled at 44.1 kHz, then then is less than 1 second of data.

$N^2 \approx 1$ billion

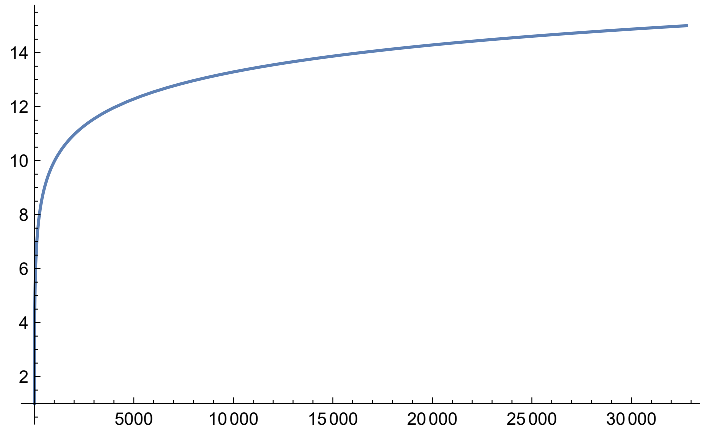
$N \log_2(N) \approx 50,\ 000$

A different of a factor of more than 2000

```
In[∘]:= Nd = 2^15
        Nd^2
        Nd Log[2 , Nd] // N
             Nd^2
        ─────────────── // N
        Nd Log[2 , Nd]

Out[∘]= 32 768

Out[∘]= 1 073 741 824

Out[∘]= 491 520.

Out[∘]= 2184.53
```

This is because the term $\log_2(n)$ starts to flatten as N increases.

In[ ]:=
```
Plot[Log[2, Nn], {Nn, 2^1, 2^15}, PlotRange → All]
```

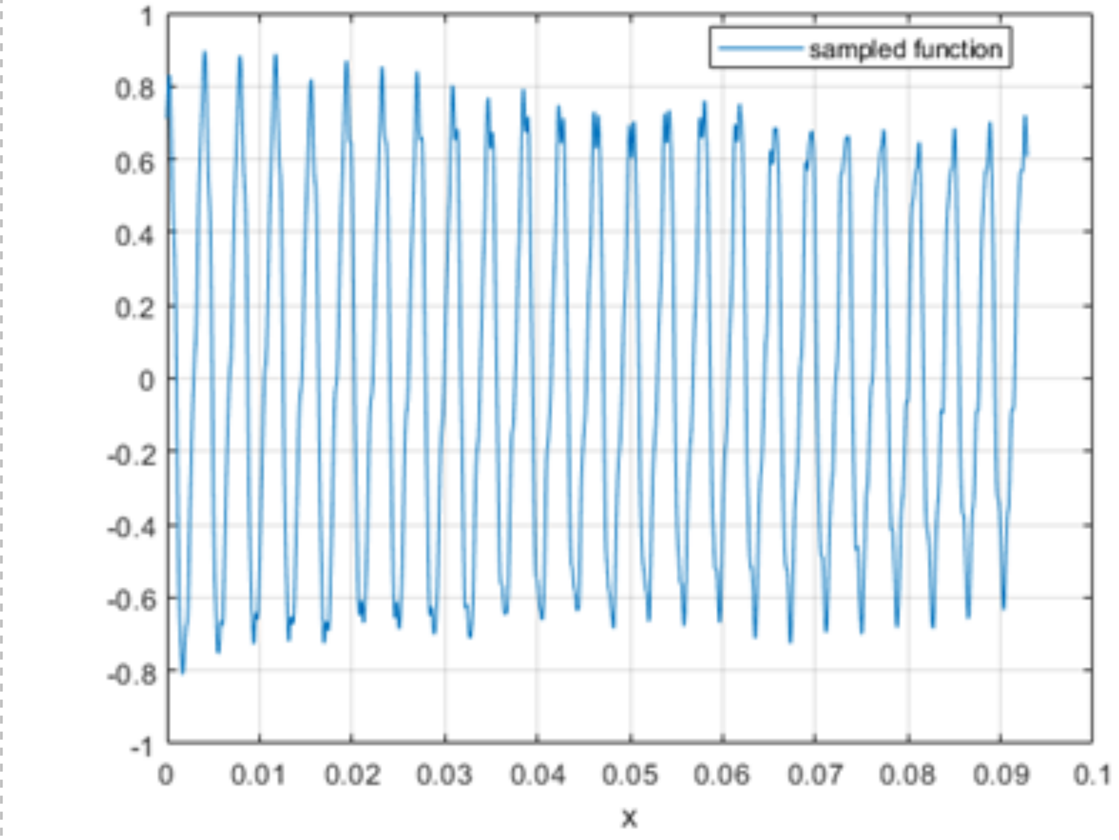
Out[ ]=


Example in Matlab

Matlab implements the FFT algorithm using a function called ‘fft’. We can demonstrate these savings with an example in Matlab.

method A:	Direction calculation using Eq.1.1
method B:	DFT matrix using Eq.1.3
method C:	Matlab’s fft

Below is an example using 2^{12} = 4096 point of audio signal sampled at 44.1 kHz of someone playing a piano middle C note

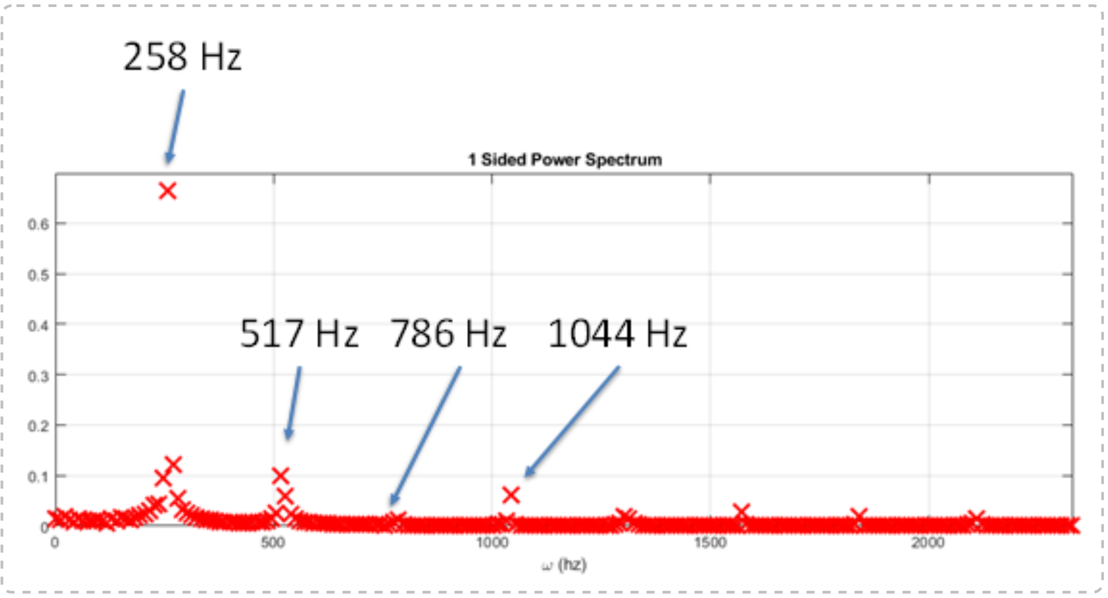


Performing the DFT using the aforementioned method takes the following amount of time

method A ≈ 14.2 sec
method B ≈ 14.0 sec
method C ≈ 0.0001 sec

So in this case Matlab’s fft is almost 90,000 times faster than the manual method.

The 1-sided power spectrum is shown below



This correlates with the fact that the middle C on a piano is supposed to be at 256 Hz (perhaps this piano is a little out of tune).