# DFT Implementation

Narendiran S

19-06-2021

The DFT equation is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$$

Implementations were done using the following methods:

a) For Loops

b) `dftmtx`

c) Vector Multiplications

d) Decimation In Time (DIT) (Cooley-Tueky)

## 1 Using For Loops

Steps:

- Varry k from 0 . . . N-1 to find each value of X.
- Varry n from 0 . . . N-1 to find individual value of X[k].
- Find the exponential argument $\frac{2\pi}{N}kn$.
- Apply exponential on this argument.
- Multiply this exponential value with x[n].
- Accumulate this value for n ranging from 0 . . . N-1.
- Store the results of accumulated value in corresponding X[k]

## 2 Using ***dftmtx***

Using the inbuilt *dftmtx* function in Matlab. *dftmtx(N)* gives the F matrix required to solve for DFT from the input.

The basic equation helps in solving this is:

$$\begin{pmatrix} X[0] \\ X[1] \\ X[2] \\ \vdots \\ X[N-1] \end{pmatrix} = \begin{pmatrix} W_N^{0.0} & W_N^{0.1} & W_N^{0.2} & \dots & W_N^{0.(N-1)} \\ W_N^{1.0} & W_N^{1.1} & W_N^{1.2} & \dots & W_N^{1.(N-1)} \\ W_N^{2.0} & W_N^{2.1} & W_N^{2.2} & \dots & W_N^{2.(N-1)} \\ \vdots \\ W_N^{(N-1).0} & W_N^{(N-1).1} & W_N^{(N-1).2} & \dots & W_N^{(N-1).(N-1)} \end{pmatrix} \begin{pmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{pmatrix}$$

$$X = Fx$$

## 3 Vector Multiplication

By creating k and n vectors and using them to instead of for loops. k Vector is a column vector of $1 \times N$.

$$kVector = \begin{pmatrix} 0 & 1 & 2 & \dots & N-2 & N-1 \end{pmatrix}$$

nVector is a row vector of $N \times 1$.

$$nVector = \begin{pmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ N-2 \\ N-1 \end{pmatrix}$$

We perform multiplication to get the Time-Frequency matrix:

$$Matrix = nVector * kVector$$

$$= \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 2 & \dots & N-2 & N-1 \\ 0 & 2 & 4 & \dots & 2*(N-2) & 2*(N-1) \\ \vdots \\ 0 & (N-1)*1 & (N-1)*2 & \dots & (N-1)*(N-2) & (N-1)*(N-1) \end{pmatrix}$$

We perform exponential on this matrix:

$$ExpMatrix = e^{-j\frac{2\pi}{N}Matrix}$$

Finally We perform Dot product of input x with the ExpMatrix to get the DFT output X.

$$X = x.ExpMatrix$$

The matrix Implementation is shown to understand this.

```
clc;
clear;
close all;
x = [1,3,4,5];

X = zeros(size(x));

N = length(x);


kVector = 0 : 1 : N-1;
nVector = 0 : 1 : N-1;
nVector = nVector';

Mat = kVector.*nVector;
ExpMat = exp(-j*Mat*2*pi/N);

X = x * ExpMat;

disp(X);
```

## 4 Decimation In Time

Only for N in powers of 2. If not in powers of 2, append zeros to the right of input. Better then vectorized and inbuilt fft (numpy) avialble in python.

We split into even and odd part. (Reduces the N-DFT $\frac{N}{2}$ DFT) Then multiply odd part with $W_N$. Finally add the reuslt with the even part to get results. Recusion is used to get $\frac{N}{2}$ DFTs. 2-point DFT is a simple addition and subtraction as shown below which is used as the end condition of recursion.

The python implementation helps in understanding the working.

```
def DFT2Point(xin):
    return np.asarray([xin[0]+xin[1], xin[0]-xin[1]])


def DFTUsingDITRecursive_2powers(xin):
    N = xin.shape[0]
    if np.log2(N) % 1 > 0:
```

```python
            print('N must powers of 2')
            return
        if N == 2:
            return DFT2Point(xin)

        xeven = xin[0::2]
        xodd = xin[1::2]

        Xeven = DFTUsingDITRecursive_2powers(xeven)
        Xodd = DFTUsingDITRecursive_2powers(xodd)

        WN = np.exp(-1j * (2*np.pi/N) * np.arange(0, N))

        # multiply with only first half of WN - this is all it's needed
        XoddM = np.multiply(Xodd, WN[ : N//2])

        Temp1 = Xeven + XoddM
        Temp2 = Xeven - XoddM
        return np.hstack((Temp1, Temp2))
```