Real forward and inverse FFT

by

Jens Hee
http://jenshee.dk

March 2014

Change log

14. march 2014

1. Document started.

Contents

Chapter 1

The Discrete Fourier Transform

1.1 Definition of the Discrete Fourier transform

The definition of the Discrete Fourier transform (DFT) used in the following is given by:

$$X(k) = DFT_N\{x(n)\} = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, \qquad k = 0, ..., N-1$$

The Inverse Discrete Fourier transform (IDFT) is defined by:

$$x(n) = IDFT_N\{X(k)\} = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}nk}$$

IDFT can be calculated from the DFT as follows:

$$x(n) = IDFT_N\{X(k)\} = \frac{1}{N} DFT_N\{X^*(k)\}^*$$

Chapter 2

The Discrete Fourier Transform of real valued signals

2.1 Some definitions

In the next sections the following definitions are used:

$x_1(n) = x(2n)$
$x_2(n) = x(2n+1)$
$X_1(k) = DFT_{N/2}\{x_1(n)\}$
$X_2(k) = DFT_{N/2}\{x_2(n)\}$
$y(n) = x_1(n) + jx_2(n)$
$Y(k) = DFT_{N/2}\{y(n)\}$

2.2 Forward DFT of a real values signal

Given a real valued signal $x(n)$, $X(k)$ can be calculated as:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn} \\ &= \sum_{n=0}^{N/2-1} x(2n) e^{-j\frac{2\pi}{N}k2n} + \sum_{n=0}^{N/2-1} x(2n+1) e^{-j\frac{2\pi}{N}k(2n+1)} \\ &= \sum_{n=0}^{N/2-1} x_1(n) e^{-j\frac{2\pi}{N/2}kn} + \sum_{n=0}^{N/2-1} x_2(n) e^{-j\frac{2\pi}{N/2}kn} e^{-j\frac{2\pi}{N}k} \\ &= X_1(k) + X_2(k) e^{-j\frac{2\pi}{N}k} \end{aligned}$$

Since x_1 and x_2 are real valued:

$$\begin{aligned} Y(k) &= X_1 + jX_2 \\ Y_{re} &= X_{1r} \\ Y_{ro} &= -X_{2i} \\ Y_{ie} &= X_{2r} \\ Y_{io} &= X_{1i} \end{aligned}$$

where:
Y_{re} is the even part of the real part of Y
Y_{ro} is the odd part of the real part of Y

Y_{ie} is the even part of the imaginary part of Y
Y_{io} is the odd part of the imaginary part of Y

$X(k)$ is finally given by:

$$\begin{aligned}
X(k) &= X_{1r}(k) + jX_{1i}(k) + (X_{2r}(k) + jX_{2i})e^{-j\frac{2\pi}{N}k} \\
&= Y_{re}(k) + jY_{io}(k) + (Y_{ie}(k) - jY_{ro}(k))e^{-j\frac{2\pi}{N}k} \\
&= 0.5(Y_r(k) + Y_r(N/2-k) + j(Y_i(k) - Y_i(N/2-k)) \\
&\quad +(Y_i(k) + Y_i(N/2-k) - j(Y_r(k) - Y_r(N/2-k)))e^{-j\frac{2\pi}{N}k}) \\
&= 0.5((Y_r(k) + jY_i(k))(1 - je^{-j\frac{2\pi}{N}k}) + (Y_r(N/2-k) - jY_i(k))(1 + je^{-j\frac{2\pi}{N}k})) \\
&= Y(k)A(k) + Y^*(N/2-k)B(k)
\end{aligned}$$

where:
$A(k) = 0.5(1 - je^{j\frac{-2\pi}{N}k})$
$B(k) = 0.5(1 + je^{j\frac{-2\pi}{N}k})$

Note:
$X(0) = Y(0)(1-j) + Y^*(N/2)(1+j) = Y_r(0) + Y_i(0)$
$X(N/2) = Y(N/2)(1+j) + Y^*(0)(1-j) = Y_r(0) - Y_i(0)$

2.3 Real DFT step by step

1. Form the complex signal $y(n)$. This is straight forward since the complex input to an FFT is often ordered as pairs of real and imaginary values, so no reordering of the input is necessary.
2. Do an N/2-point DFT using an N/2-point FFT.
3. Calculate the two arrays $A(k)$ and $B(k)$ for $0 \leq k < N/2$.
4. Combine the spectral values $Y(k)$ with $A(k)$ and $B(k)$ to form $X(k)$ for $0 \leq k < N/2$. Usualy $X(k)$ is zero at half the samplig frequency and $X(N/2)$ need not be calculated.

2.4 Inverse DFT given a real values signal

Given a spectrum $X(k)$ of a real valued signal $x(n)$, $y(n)$ can be calculated as:

$$\begin{aligned}
y(n) &= x(2n) + jx(2n+1) \\
&= \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi}{N}2nk} + j\frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi}{N}(2n+1)k} \\
&= \frac{1}{N}\sum_{k=0}^{N-1} X(k)(1 + je^{j\frac{2\pi}{N}k})e^{j\frac{2\pi}{N/2}nk} \\
&= \frac{1}{N}\sum_{k=0}^{N/2-1} X(k)(1 + je^{j\frac{2\pi}{N}k})e^{j\frac{2\pi}{N/2}nk} + \frac{1}{N}\sum_{k=0}^{N/2-1} X(k+N/2)(1 + je^{j\frac{2\pi}{N}(k+N/2)})e^{j\frac{2\pi}{N/2}n(k+N/2)} \\
&= \frac{1}{N}\sum_{k=0}^{N/2-1} X(k)(1 + je^{j\frac{2\pi}{N}k})e^{j\frac{2\pi}{N/2}nk} + \frac{1}{N}\sum_{k=0}^{N/2-1} X(k+N/2)(1 - je^{j\frac{2\pi}{N}k})e^{j\frac{2\pi}{N/2}nk}
\end{aligned}$$

Since $X(k) = X^*(N-k)$ we have:

$$\begin{aligned} y(n) &= \frac{1}{N/2}\sum_{k=0}^{N/2-1} 0.5X(k)(1+je^{j\frac{2\pi}{N}k})e^{j\frac{2\pi}{N/2}nk} + \frac{1}{N/2}\sum_{k=0}^{N/-1} 0.5X^*(N/2-k)(1-je^{j\frac{2\pi}{N}k})e^{j\frac{2\pi}{N/2}nk} \\ &= \frac{1}{N/2}\sum_{k=0}^{N/2-1} (X(k)A^*(k) + X^*(N/2-k)B^*(k))e^{j\frac{2\pi}{N/2}nk} \\ Y(k) &= X(k)A^*(k) + X^*(N/2-k)B^*(k) \end{aligned}$$

where A and B are definded above.

Note:
$Y(0) = 0.5(1+j)X(0) + 0.5(1-j)X(N/2)$

2.5 Real IDFT step by step

1. Calculate the two arrays $A(k)$ and $B(k)$ for $0 \leq k < N/2$.
2. Combine the spectral values $X(k)$ with $A(k)$ and $B(k)$ to form $Y(k)$ for $0 \leq k < N/2$. Usualy $X(k)$ is zero at half the samplig frequency and the last term in $Y(0)$ need not be taken into account.
3. Do an N/2-point IDFT of $Y(k)$ using an N/2-point IFFT.
4. If the complex input and output to the FFT is ordered as pairs of real and imaginary values, then no reordering of the output is necessary.

Chapter 3

Program examples

```
#include "math.h"

static double pi = 4.0 * atan(1.0);

void FFT(int M, double* X)
{
const int N = (int)floor(pow(2.0, M) + 0.5);

double wcos;
double wsin;
double ucos;
double usin;

int N2 = N >> 1;
int L1 = N;
int L2;

for (int L = 0; L < M; L++)
{
L2 = L1 >> 1;

wcos = 1;
wsin = 0;
ucos = cos(pi / L2);
usin = -sin(pi / L2);

for (int J = 0; J < L2; J++)
{
for (int I = J; I < N; I = I + L1)
{
int Ir = 2 * I;
int Ii = Ir + 1;
int L22 = 2 * L2;

double sumRe = X[Ir] + X[Ir + L22];
double sumIm = X[Ii] + X[Ii + L22];

double diffRe = X[Ir] - X[Ir + L22];
double diffIm = X[Ii] - X[Ii + L22];

X[Ir + L22] = diffRe * wcos - diffIm * wsin;
X[Ii + L22] = diffRe * wsin + diffIm * wcos;

X[Ir] = sumRe;
X[Ii] = sumIm;
}

double w = wcos * ucos - wsin * usin;
wsin = wcos * usin + wsin * ucos;
wcos = w;
}
L1 = L1 >> 1;
}

int K;
int J = 0;
int N1 = N - 1;
```

```
for (int I = 0; I < N1; I++)
{
if (I < J)
{
int Jr = 2 * J;
int Ji = Jr + 1;
int Ir = 2 * I;
int Ii = Ir + 1;
double TRe = X[Jr];
double TIm = X[Ji];
X[Jr] = X[Ir];
X[Ji] = X[Ii];
X[Ir] = TRe;
X[Ii] = TIm;
}
K = N2;
while (K <= J)
{
J = J - K;
K = K >> 1;
}
J = J + K;
}
}

void IFFT(int M, double* X)
{
const int N = (int)floor(pow(2.0, M) + 0.5);

double wcos;
double wsin;
double ucos;
double usin;

int N2 = N >> 1;
int L1 = N;
int L2;

for (int L = 0; L < M; L++)
{
L2 = L1 >> 1;

wcos = 1;
wsin = 0;
ucos = cos(pi / L2);
usin = sin(pi / L2);

for (int J = 0; J < L2; J++)
{
for (int I = J; I < N; I = I + L1)
{
int Ir = 2 * I;
int Ii = Ir + 1;
int L22 = 2 * L2;

double sumRe = 0.5 * (X[Ir] + X[Ir + L22]);
double sumIm = 0.5 * (X[Ii] + X[Ii + L22]);

double diffRe = 0.5 * (X[Ir] - X[Ir + L22]);
double diffIm = 0.5 * (X[Ii] - X[Ii + L22]);

X[Ir + L22] = diffRe * wcos - diffIm * wsin;
X[Ii + L22] = diffRe * wsin + diffIm * wcos;

X[Ir] = sumRe;
X[Ii] = sumIm;
}

double w = wcos * ucos - wsin * usin;
wsin = wcos * usin + wsin * ucos;
wcos = w;
}
L1 = L1 >> 1;
}

int K;
```

```
int J = 0;
int N1 = N - 1;

for (int I = 0; I < N1; I++)
{
if (I < J)
{
int Jr = 2 * J;
int Ji = Jr + 1;
int Ir = 2 * I;
int Ii = Ir + 1;
double TRe = X[Jr];
double TIm = X[Ji];
X[Jr] = X[Ir];
X[Ji] = X[Ii];
X[Ir] = TRe;
X[Ii] = TIm;
}
K = N2;
while (K <= J)
{
J = J - K;
K = K >> 1;
}
J = J + K;
}
}

void RealFFT(int M, double* X)
{
FFT(M - 1, X);

int N = (int)floor(pow(2.0, M) + 0.5);

double* A = new double[N];
double* B = new double[N];

for (int i = 0; i < N / 2; i++)
{
A[2*i] = 0.5 * (1 - sin(2 * pi / N * i));
A[2*i+1] = -0.5 * cos(2 * pi / N * i);
B[2*i] = 0.5 * (1 + sin(2 * pi / N * i));
B[2*i+1] = 0.5 * cos (2 * pi / N * i);
}

for (int k = 1; k < N / 4 + 1; k++)
{
int k_2 = 2 * k;

double xr = X[k_2] * A[k_2] - X[k_2+1] * A[k_2+1] + X[N-k_2] * B[k_2] + X[N-k_2+1] * B[k_2+1];
double xi = X[k_2] * A[k_2+1] + X[k_2+1] * A[k_2] + X[N-k_2] * B[k_2+1] - X[N-k_2+1] * B[k_2];
double xrN = X[N-k_2] * A[N-k_2] - X[N-k_2+1] * A[N-k_2+1] + X[k_2] * B[N-k_2] + X[k_2+1] * B[N-k_2+1];
double xiN = X[N-k_2] * A[N-k_2+1] + X[N-k_2+1] * A[N-k_2] + X[k_2] * B[N-k_2+1] - X[k_2+1] * B[N-k_2];
X[k_2] = xr;
X[k_2+1] = xi;
X[N-k_2] = xrN;
X[N-k_2+1] = xiN;
}

double temp = X[0];
X[0]= X[0] + X[1];
X[N]= temp - X[1];
X[1]= 0;
X[N+1] = 0;
}

void RealIFFT(int M, double* X)
{
int N = (int)floor(pow(2.0, M) + 0.5);

double* A = new double[N];
double* B = new double[N];

for (int i = 0; i < N / 2; i++)
{
A[2*i] = 0.5 * (1 - sin(2 * pi / N * i));
A[2*i+1] = -0.5 * cos(2 * pi / N * i);
```

```
B[2*i] = 0.5 * (1 + sin(2 * pi / N * i));
B[2*i+1] = 0.5 * cos (2 * pi / N * i);
}

for (int k = 1; k < N / 4 + 1; k++)
{
int k_2 = 2 * k;

double xr = X[k_2] * A[k_2] + X[k_2+1] * A[k_2+1] + X[N-k_2] * B[k_2] - X[N-k_2+1] * B[k_2+1];
double xi  = -X[k_2] * A[k_2+1] + X[k_2+1] * A[k_2] - X[N-k_2] * B[k_2+1] - X[N-k_2+1] * B[k_2];
double xrN = X[N-k_2] * A[N-k_2] + X[N-k_2+1] * A[N-k_2+1] + X[k_2] * B[N-k_2] - X[k_2+1] * B[N-k_2+1];
double xiN = -X[N-k_2] * A[N-k_2+1] + X[N-k_2+1] * A[N-k_2] - X[k_2] * B[N-k_2+1] - X[k_2+1] * B[N-k_2];
X[k_2] = xr;
X[k_2+1] = xi;
X[N-k_2] = xrN;
X[N-k_2+1] = xiN;
}
double temp = X[0];
X[0]= 0.5 * X[0] + 0.5 * X[N];
X[1]= 0.5 * temp - 0.5 * X[N];

IFFT(M - 1, X);
}
```