# HW6 Program Assignment

**By: 105060012** 張育菘

## P1

Consider again the inverse DFT given in (8.2).

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn},\ n = 0, 1, \ldots, N-1 \quad (8.2)$$

(a) Replace k by $\langle -k\rangle_N$ in (8.2) and show that the resulting summation is a DFT expression, that is, $\mathrm{IDFT}\left\{X\left[k\right]\right\} = \frac{1}{N}\mathrm{DFT}\{X[\langle -k\rangle_N]\}$.

Ans.

$$\mathrm{IDFT}\left\{X\left[k\right]\right\} = x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn}$$

$$\mathrm{DFT}\{X[\langle -k\rangle_N]\} = \sum_{k=0}^{N-1} X[\langle -k\rangle_N]W_N^{kn} = \sum_{k=0}^{N-1} X[k]W_N^{-kn}$$

$$\Longrightarrow \mathrm{IDFT}\left\{X\left[k\right]\right\} = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn} = \frac{1}{N}\mathrm{DFT}\{X[\langle -k\rangle_N]\}$$

(b) Develop a MATLAB function x = IDFT(X,N) using the fft function that uses the above approach. Verify your function on signal x[n] = {1, 2, 3, 4, 5, 6, 7, 8}.

```
close all; clear;
fprintf('1(b)\n');
```

```
1(b)
```

```

open IDFT.m;

x = [1 2 3 4 5 6 7 8]
```

```
x = 1×8
     1     2     3     4     5     6     7     8
```

```
X = fft(x); N = length(X);
x_verify = IDFT(X, N)
```

```
x_verify = 1×8 complex
```

```
    1.0000 + 0.0000i    2.0000 + 0.0000i    3.0000 + 0.0000i    4.0000 + 0.0000i ···
```

(Comment)

由上面的結果顯示，從自己設計的 IDFT 把經由 fft 轉換得到的 X 轉回去 time domain 的 x_verify 與原本的 x 相同。

## P2

In this problem we will investigate differences in the speeds of DFT and FFT algorithms when stored twiddle factors are used.

$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \\ X[4] \\ X[5] \\ X[6] \\ X[7] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_8 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ 1 & W_8^2 & W_8^4 & W_8^6 & 1 & W_8^2 & W_8^4 & W_8^6 \\ 1 & W_8^3 & W_8^6 & W_8^1 & W_8^4 & W_8^7 & W_8^2 & W_8^5 \\ 1 & W_8^4 & 1 & W_8^4 & 1 & W_8^4 & 1 & W_8^4 \\ 1 & W_8^5 & W_8^2 & W_8^7 & W_8^4 & W_8 & W_8^6 & W_8^3 \\ 1 & W_8^6 & W_8^4 & W_8^2 & 1 & W_8^6 & W_8^4 & W_8^2 \\ 1 & W_8^7 & W_8^6 & W_8^5 & W_8^4 & W_8^3 & W_8^2 & W_8 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \\ x[4] \\ x[5] \\ x[6] \\ x[7] \end{bmatrix} \quad (8.8)$$

(a) Write a function W = dft_matrix(N) that computes the DFT matrix $W_N$ given in (8.8).

Ans.

$$W_N = e^{-j\frac{2\pi}{N}},\ (W_N)_{(n+1)\times(m+1)} = \left(e^{-j\frac{2\pi}{N}}\right)^{nm},\ n, m = 0, 1, \ldots, N-1$$

```
close all; clear;
fprintf('2(a)\n');
```

```
2(a)
```

```
open dft_matrix.m;
```

(b) Write a function X = dftdirect_m(x,W) that modifies the dftdirect function using the matrix W from (a). Using the tic and toc functions compare computation times for the dftdirect and dftdirect_m function for N = 128, 256, 512, and 1024. For this purpose generate an N-point complex-valued signal as x = randn(1,N) + 1j*randn(1,N). (verify your code with fft first)

```
close all; clear;
fprintf('2(b)\n');
```

```
2(b)
```

```
open dftdirect_m.m;

N_set = [128 256 512 1024];
for i = 1:1:length(N_set)
    fprintf('N = %d\n', N_set(i));
    x = randn(1,N_set(i)) + 1j*randn(1,N_set(i));

    fprintf('  by dftdirect_m\n');
    tic;
    W = dft_matrix(N_set(i));
    X1 = dftdirect_m(x, W)
    toc;

    fprintf('  by dftdirect\n');
    tic;
    X2 = dftdirect(x)
    toc;
end
```

```
N = 128
  by dftdirect_m
X1 = 1×128 complex
  14.8600 -16.9911i   9.9053 -18.4707i  12.4703 - 8.8463i -11.5983 -11.2830i ⋯
Elapsed time is 0.036558 seconds.
  by dftdirect
X2 = 1×128 complex
  14.8600 -16.9911i   9.9053 -18.4707i  12.4703 - 8.8463i -11.5983 -11.2830i ⋯
Elapsed time is 0.105756 seconds.
N = 256
  by dftdirect_m
X1 = 1×256 complex
  -9.3494 -10.8823i  27.1163 + 7.5972i  12.2987 - 2.1780i  13.4434 +26.4978i ⋯
Elapsed time is 0.057577 seconds.
  by dftdirect
X2 = 1×256 complex
  -9.3494 -10.8823i  27.1163 + 7.5972i  12.2987 - 2.1780i  13.4434 +26.4978i ⋯
Elapsed time is 0.094122 seconds.
N = 512
  by dftdirect_m
X1 = 1×512 complex
  26.0656 - 3.7932i -19.1526 +43.1327i  41.4818 +33.4331i  28.1913 -13.2635i ⋯
Elapsed time is 0.215067 seconds.
  by dftdirect
X2 = 1×512 complex
  26.0656 - 3.7932i -19.1526 +43.1327i  41.4818 +33.4331i  28.1913 -13.2635i ⋯
Elapsed time is 0.328506 seconds.
N = 1024
  by dftdirect_m
X1 = 1×1024 complex
10^2 ×
   0.3573 + 0.3994i   0.1048 - 0.2627i  -0.6048 + 0.2213i   0.1171 + 0.3169i ⋯
Elapsed time is 0.719672 seconds.
  by dftdirect
```

```
X2 = 1×1024 complex
10^2 ×
   0.3573 + 0.3994i   0.1048 - 0.2627i  -0.6048 + 0.2213i   0.1171 + 0.3169i ···
Elapsed time is 1.553048 seconds.
```

Ans:

由output結果可得知，x 經過 W = dft_matrix(N) 和 X = dftdirect_m(x,W) 得到 X 的速度比直接做 DFT (X = dftdirect(x)) 快，這是因為 dftdirect 是把所有 W*x 的 term 算完後再做加總；而前者的方法是將 W 的 matrix 算完作為 input 至 dftdirect_m 得到 X，因此此方法會比較快。

(c) Write a function X = fftrecur_m(x,W) that modifies the fftrecur function given on page 439 using the matrix W from (a). Using the tic and toc functions compare computation times for the fftrecur and fftrecur_m function for N = 128, 256, 512, and 1024. For this purpose generate an N-point complex valued signal as x = randn(1,N) + 1j*randn(1,N). (verify your code with fft first)

Ans.

$$X = W_8 x = W_4 \begin{bmatrix} I & D_8 \\ I & -D_8 \end{bmatrix} \begin{bmatrix} x_E \\ x_O \end{bmatrix} = \begin{bmatrix} I & D_8 \\ I & -D_8 \end{bmatrix} \begin{bmatrix} W_4 x_E \\ W_4 x_O \end{bmatrix},$$

$$W_4 x_E = W_2 \begin{bmatrix} I & D_4 \\ I & -D_4 \end{bmatrix} \begin{bmatrix} x_{\mathrm{EE}} \\ x_{\mathrm{EO}} \end{bmatrix} = \begin{bmatrix} I & D_4 \\ I & -D_4 \end{bmatrix} \begin{bmatrix} W_2 x_{\mathrm{EE}} \\ W_2 x_{\mathrm{EO}} \end{bmatrix},$$

$$W_2 x_{\mathrm{EE}} = W_1 \begin{bmatrix} I & D_2 \\ I & -D_2 \end{bmatrix} \begin{bmatrix} x_{\mathrm{EEE}} \\ x_{\mathrm{EEO}} \end{bmatrix} = \begin{bmatrix} I & D_2 \\ I & -D_2 \end{bmatrix} \begin{bmatrix} x_{\mathrm{EEE}} \\ x_{\mathrm{EEO}} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{EEE}} \\ x_{\mathrm{EEO}} \end{bmatrix}$$

```
close all; clear;
fprintf('2(c)\n');
```

```
2(c)
```

```

open fftrecur_m.m;

N_set = [128 256 512 1024];
for i = 1:1:length(N_set)
    fprintf('N = %d\n', N_set(i));
    x = randn(1,N_set(i)) + 1j*randn(1,N_set(i));

    fprintf('  by fftrecur_m\n');
    tic;
    W = dft_matrix(N_set(i));
    X1 = fftrecur_m(x, W)
    toc;

    fprintf('  by fftrecur\n');
    tic;
    X2 = fftrecur(x)'
    toc;
```

```
end
```

```
N = 128
  by fftrecur_m
X1 = 128×1 complex
  -0.6522 - 5.2027i
  -2.9417 + 1.7876i
   8.5636 -18.7036i
  -6.6545 +18.0341i
  -5.6074 + 2.7283i
  -3.3954 + 1.2980i
   4.2204 +10.6196i
   5.6173 +21.1089i
  12.4469 + 3.3469i
  -3.7409 + 2.2237i
      ⋮

Elapsed time is 0.031175 seconds.
  by fftrecur
X2 = 1×128 complex
  -0.6522 + 5.2027i   4.9469 - 4.3094i  13.1789 -15.9463i -12.2957 +12.7107i ⋯
Elapsed time is 0.017077 seconds.
N = 256
  by fftrecur_m
X1 = 256×1 complex
  -2.9769 +20.1662i
   3.0187 - 7.2361i
  10.1548 - 7.3545i
   5.9500 +31.4920i
 -14.5183 -24.1378i
   2.4679 +22.1131i
  -2.7589 - 1.6438i
 -20.6436 + 3.0748i
  18.6272 + 2.7847i
  -1.1683 +33.7523i
      ⋮

Elapsed time is 0.040165 seconds.
  by fftrecur
X2 = 1×256 complex
  -2.9769 -20.1662i  -3.1185 -28.9135i  16.7614 +11.1979i  -9.5674 +50.6165i ⋯
Elapsed time is 0.062551 seconds.
N = 512
  by fftrecur_m
X1 = 512×1 complex
  22.2150 -24.7398i
   0.5101 +16.0453i
  -6.7364 - 1.8906i
 -28.1948 +13.8535i
   8.6954 -20.6527i
 -27.8050 -16.1379i
  19.3390 +19.3092i
  26.5463 + 9.4375i
  -4.6983 -36.3568i
  -5.1256 -16.3752i
      ⋮

Elapsed time is 0.126839 seconds.
  by fftrecur
X2 = 1×512 complex
```

```
    22.2150 +24.7398i  17.0689 -11.9182i   8.5055 - 3.2886i  32.0359 -27.7643i ···
  Elapsed time is 0.004490 seconds.
  N = 1024
    by fftrecur_m
  X1 = 1024×1 complex
  10^2 ×
    -0.2566 - 0.1289i
    -0.2993 + 0.4810i
    -0.0705 + 0.4556i
    -0.2974 + 0.1410i
    -0.1280 - 0.1023i
    -0.0213 + 0.3705i
     0.0716 - 0.4301i
    -0.1349 - 0.1314i
    -0.1056 + 0.1588i
    -0.0822 + 0.3717i
        ⋮
  Elapsed time is 0.499928 seconds.
    by fftrecur
  X2 = 1×1024 complex
  10^2 ×
    -0.2566 + 0.1289i   0.1120 + 0.1124i   0.2518 + 0.5606i  -0.3665 - 0.1495i ···
  Elapsed time is 0.011612 seconds.
```

由output結果可得知，x 經過 fftrecur_m 得到 X 的速度比做 fftrecur 慢，這是因為 fftrecur 是直接算所需的 w；而前者的方法在每次迴圈都會把 W 的 matrix 作為 input 至下一層的 fftrecur_m ，等於是在每次迴圈都會傳一些不會用到的資料，因此此方法會比較慢。

## P3

Consider the flow graph in Figure 8.10 which implements a DIT-FFT algorithm with both input and output in natural order. Let the nodes at each stage be labeled as sm[k],0 ≤ m ≤ 3 with s0[k] = x[k] and s3[k] = X[k], 0 ≤ k ≤ 7.

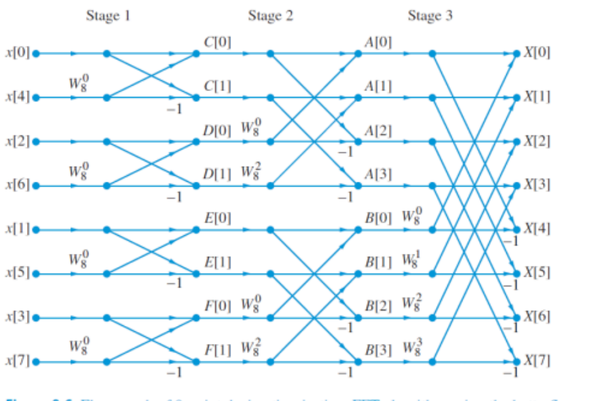


**Figure 8.6** Flow graph of 8-point decimation-in-time FFT algorithm using the butterfly computation shown in Figure 8.4. The trivial twiddle factor $W_8^0 = 1$ is shown for the sake of generality.

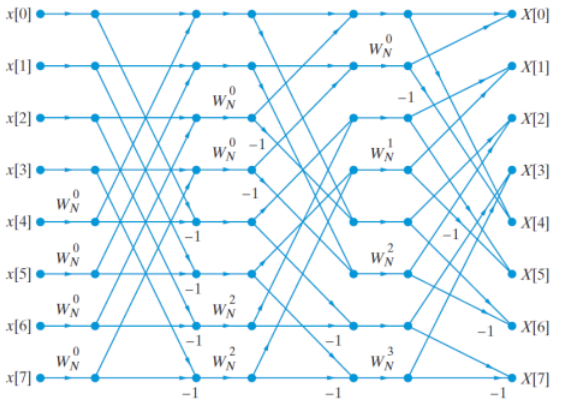


**Figure 8.10** Decimation-in-time FFT algorithm with both input and output in natural order.

(a) Express $s_m[k]$ in terms of $s_{m-1}[k]$ for m = 1, 2, 3.

Ans.

$s_1[\mathrm{k1}] = x[\mathrm{k1}] + x[\mathrm{k1}+4], s_1[\mathrm{k1}+4] = x[\mathrm{k1}] - x[\mathrm{k1}+4], \mathrm{k1} = 0, 1, 2, 3$

$s_2[\mathrm{k2}] = s_1[\mathrm{k2}] + s_1[\mathrm{k2}+2], s_2[\mathrm{k2}+2] = s_1[\mathrm{k2}+4] + W_8^2 s_1[\mathrm{k2}+6],$

$s_2[\mathrm{k2}+4] = s_1[\mathrm{k2}] - s_1[\mathrm{k2}+2], s_2[\mathrm{k2}+6] = s_1[\mathrm{k2}+4] - W_8^2 s_1[\mathrm{k2}+6],\ \mathrm{k2} = 0, 1$

$s_3[0] = s_2[0] + s_2[1], s_3[1] = s_2[2] + W_8^1 s_2[3], s_3[2] = s_2[4] + W_8^2 s_2[5], s_3[3] = s_2[6] + W_8^3 s_2[7]$

$s_3[4] = s_2[0] - s_2[1], s_3[5] = s_2[2] - W_8^1 s_2[3], s_3[6] = s_2[4] - W_8^2 s_2[5], s_3[7] = s_2[6] - W_8^3 s_2[7]$

(b) Write a MATLAB function X = fftalt8(x) that computes an 8-point DFT using the equations in part (a). Verify with sequence x[n] = {0,1,2,2,3,3,3,4}.

```
close all; clear;
fprintf('3(b)\n');
```

```
3(b)
```

```

open fftalt8.m;

x = [0 1 2 2 3 3 3 4];
X_fft = fft(x,8)
```

```
X_fft = 1×8 complex
  18.0000 + 0.0000i  -3.0000 + 3.8284i  -2.0000 + 2.0000i  -3.0000 + 1.8284i ⋯
```

```
X_fftalt8 = fftalt8(x)
```

```
X_fftalt8 = 1×8 complex
  18.0000 + 0.0000i  -3.0000 + 3.8284i  -2.0000 + 2.0000i  -3.0000 + 1.8284i ⋯
```

(c) Compare the coding complexity of the above function with that of MATLAB function fftditr2 shown in Figure 8.6, and comment on its usefulness.

```
close all; clear;
fprintf('3(c)\n');
```

```
3(c)
```

```
x = [0 1 2 2 3 3 3 4];
X_fftditr2 = fftditr2(x)
```

```
X_fftditr2 = 1×8 complex
  18.0000 + 0.0000i  -3.0000 + 3.8284i  -2.0000 + 2.0000i  -3.0000 + 1.8284i ⋯
```

(Comment)

就 coding complexity 而言，fftalt8 顯得複雜，因為需要將前一級算完才能算下一級，而且架構相對 fftditr2 複雜，沒辦法像 fftditr2 用三層 for 迴圈就完成此 function，因次在 coding 的複雜度方面也就比 fftditr2 複雜。

## P4

Using the flow graph of Figure 8.13 and following the approach used in developing the fftditr2 function.

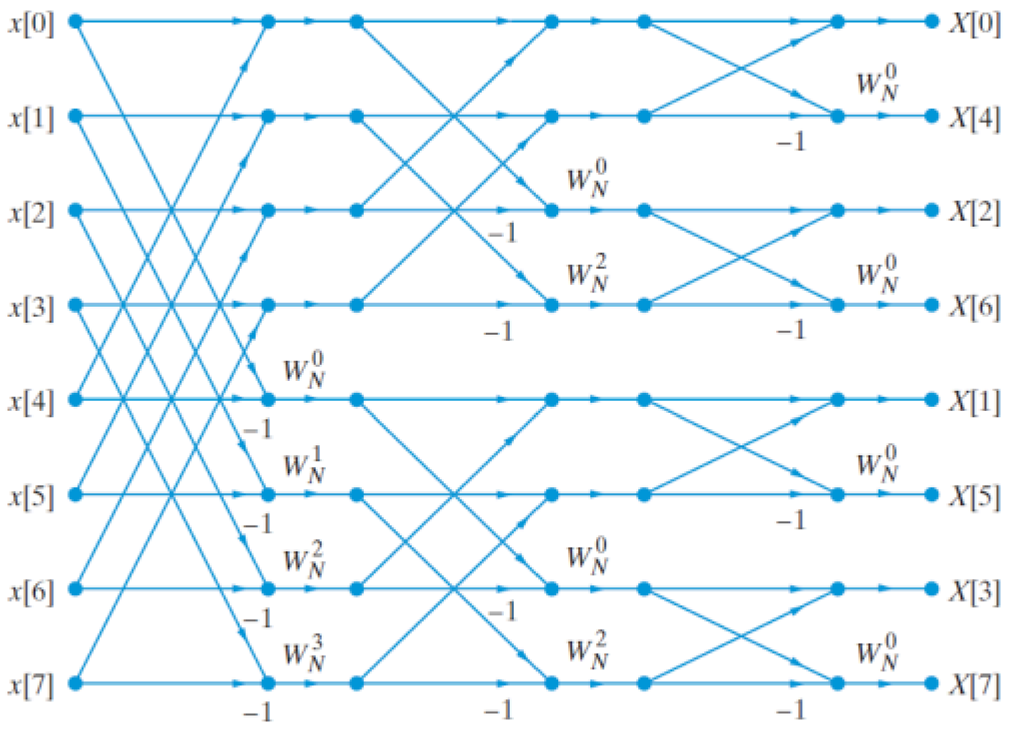


**Figure 8.13** Flow graph for the decimation-in-frequency 8-point FFT algorithm. The input sequence is in natural order and the output sequence in bit-reversed order.

(a) Develop a radix-2 DIF-FFT function X = fftdifr2(x) for power-of-2 length N.

```
close all; clear;
fprintf('4(a)\n');
```

```
4(a)
```

```

open fftdifr2.m;
```

(b) Verify your function for N = $2^\nu$, where 2 ≤ ν ≤ 10. For this purpose generate an N-point complex-valued signal as x = randn(1,N) + 1j*randn(1,N).

```
fprintf('4(b)\n');
```

```
4(b)
```

```
for v = 2:10
    N = 2^v;
    x = randn(1,N) + 1j*randn(1,N);
    fprintf('for N = 2^%d\n', v);
    X_fftdifr2 = fftdifr2(x)
    X_fft = fft(x)
end
```

```
for N = 2^2
X_fftdifr2 = 1×4 complex
   0.7229 - 0.6519i   1.2888 + 2.2962i   1.1090 - 0.8002i  -1.0473 + 1.9618i
X_fft = 1×4 complex
   0.7229 - 0.6519i   1.2888 + 2.2962i   1.1090 - 0.8002i  -1.0473 + 1.9618i
for N = 2^3
X_fftdifr2 = 1×8 complex
  -0.6388 + 1.4476i  -0.6774 + 0.2921i  -0.5768 - 2.3854i  -6.3883 - 2.8715i ⋯
X_fft = 1×8 complex
  -0.6388 + 1.4476i  -0.6774 + 0.2921i  -0.5768 - 2.3854i  -6.3883 - 2.8715i ⋯
for N = 2^4
X_fftdifr2 = 1×16 complex
  -2.8699 + 0.9639i   1.1856 + 1.0836i  -8.4793 - 2.0574i   1.2718 + 5.0888i ⋯
X_fft = 1×16 complex
  -2.8699 + 0.9639i   1.1856 + 1.0836i  -8.4793 - 2.0574i   1.2718 + 5.0888i ⋯
for N = 2^5
X_fftdifr2 = 1×32 complex
  -6.8237 - 3.9361i  -0.7731 + 3.6484i  -1.3866 - 3.3637i  -0.1640 + 2.3503i ⋯
X_fft = 1×32 complex
  -6.8237 - 3.9361i  -0.7731 + 3.6484i  -1.3866 - 3.3637i  -0.1640 + 2.3503i ⋯
for N = 2^6
X_fftdifr2 = 1×64 complex
   8.4193 + 3.7853i   1.2007 + 8.2331i   6.2577 + 2.3820i  -0.9565 - 5.1766i ⋯
X_fft = 1×64 complex
   8.4193 + 3.7853i   1.2007 + 8.2331i   6.2577 + 2.3820i  -0.9565 - 5.1766i ⋯
for N = 2^7
X_fftdifr2 = 1×128 complex
  -8.2014 - 8.0684i  -0.6122 + 2.5266i -28.2804 - 9.0278i  11.8084 + 6.1246i ⋯
X_fft = 1×128 complex
  -8.2014 - 8.0684i  -0.6122 + 2.5266i -28.2804 - 9.0278i  11.8084 + 6.1246i ⋯
for N = 2^8
X_fftdifr2 = 1×256 complex
 -21.3933 -25.0174i -10.1746 +18.1729i  -7.8854 -12.4340i  -8.2055 +37.9570i ⋯
X_fft = 1×256 complex
 -21.3933 -25.0174i -10.1746 +18.1729i  -7.8854 -12.4340i  -8.2055 +37.9570i ⋯
for N = 2^9
X_fftdifr2 = 1×512 complex
  11.7374 +19.2580i  14.1321 -13.5540i   6.5551 +36.2684i   7.7291 - 1.8899i ⋯
X_fft = 1×512 complex
  11.7374 +19.2580i  14.1321 -13.5540i   6.5551 +36.2684i   7.7291 - 1.8899i ⋯
for N = 2^10
X_fftdifr2 = 1×1024 complex
10^2 ×
   0.2291 - 0.4560i   0.0540 - 0.0919i  -0.1695 - 0.2888i  -0.2992 - 0.1905i ⋯
X_fft = 1×1024 complex
10^2 ×
   0.2291 - 0.4560i   0.0540 - 0.0919i  -0.1695 - 0.2888i  -0.2992 - 0.1905i ⋯
```

(Comment)

由上述結果我們發現 x 經過 fftdifr2 和 fft 得到的結果相同。

## P5

The filterfirdf implements the FIR direct form structure.

(a) Develop a new MATLAB function y=filterfirlp(h,x) that implements the FIR linear-phase form given its impulse response in h. This function should first check if h is one of type-I through type-IV and then simulate the corresponding equations. If h does not correspond to one of the four types then the function should display an appropriate error message.

Ams.

For Type I: $y[n] = \sum_{k=0}^{\frac{M}{2}-1} h[k](x[n-k] + x[n-M+k]) + h\left[\frac{M}{2}\right]x\left[n-\frac{M}{2}\right]$

For Type II: $y[n] = \sum_{k=0}^{\frac{M-1}{2}} h[k](x[n-k] + x[n-M+k])$

For Type III: $y[n] = \sum_{k=0}^{\frac{M}{2}-1} h[k](x[n-k] - x[n-M+k])$

For Type IV: $y[n] = \sum_{k=0}^{\frac{M-1}{2}} h[k](x[n-k] - x[n-M+k])$

```
close all; clear;
fprintf('5(a)\n');
```

```
5(a)
```

```

open filterfirlp.m;
```

(b) Verify your function on each of the following FIR systems:

h1[n] = {1,2,3,2,1}, h2[n] = {1,-2,3,3,-2,1}, h3[n] = {1,-5,0,5,-1}, h4[n] = {1,-3,-4,4,3,-1}, h5[n] = {1,2,3,-2,-1},

For verification determine the first ten samples of the step responses using your function and compare them with those from the filter function.

```
fprintf('5(b)\n');
```

```
5(b)
```

```

x = ones(1, 10);
```

```
h1 = [1 2 3 2 1];
[y1, type] = filterfirlp(h1, x)
```

```
y1 = 1×10
     1     3     6     8     9     9     9     9     9     9
type = 
'Type I'
```

```
y1_verify = filterfirdf(h1, x)
```

```
y1_verify = 1×10
     1     3     6     8     9     9     9     9     9     9
```

```
h2 = [1 -2 3 3 -2 1];
[y2, type] = filterfirlp(h2, x)
```

```
y2 = 1×10
     1    -1     2     5     3     4     4     4     4     4
type = 
'Type II'
```

```
y2_verify = filterfirdf(h2, x)
```

```
y2_verify = 1×10
     1    -1     2     5     3     4     4     4     4     4
```

```
h3 = [1 -5 0 5 -1];
[y3, type] = filterfirlp(h3, x)
```

```
y3 = 1×10
     1    -4    -4     1     0     0     0     0     0     0
type = 
'Type III'
```

```
y3_verify = filterfirdf(h3, x)
```

```
y3_verify = 1×10
     1    -4    -4     1     0     0     0     0     0     0
```

```
h4 = [1 -3 -4 4 3 -1];
[y4, type] = filterfirlp(h4, x)
```

```
y4 = 1×10
     1    -2    -6    -2     1     0     0     0     0     0
type = 
'Type IV'
```

```
y4_verify = filterfirdf(h4, x)
```

```
y4_verify = 1×10
     1    -2    -6    -2     1     0     0     0     0     0
```

```
h5 = [1 2 3 -2 -1];
[y5, type] = filterfirlp(h5, x)
```

```
y5 = 1×10
     1     3     6     4     3     3     3     3     3     3
type =
'Not correspond to one of the four types.'
```

```
y5_verify = filterfirdf(h5, x)
```

```
y5_verify = 1×10
     1     3     6     4     3     3     3     3     3     3
```

## P6

Consider the IIR normal direct form II structure given in Figure 9.6 and implemented by (9.18) and (9.20).

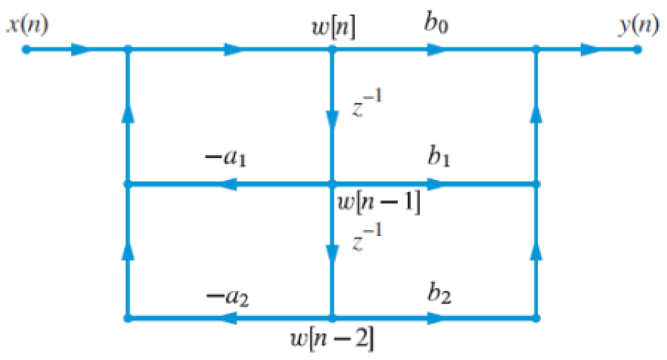


**Figure 9.6** Direct form II structure for implementation of an $N$th order system. For convenience, we assume that $N = M = 2$. If $N \neq M$, some of the coefficients will be zero.

$$y[n] = \sum_{k=0}^{M} b_k w[n-k]. \tag{9.20}$$

$$w[n] = -\sum_{k=1}^{N} a_k w[n-k] + x[n]. \tag{9.18}$$

(a) Using the MATLAB function filterdf1 as a guide, develop a MATLAB function y=filterdf2(b,a,x) that implements the normal direct form II structure. Assume zero initial conditions.

```
close all; clear;
fprintf('6(a)\n');
```

```
6(a)
```

```
open filterdf2.m;
```

(b) Determine y[n], 0 ≤ n ≤ 500 using your function and filterdf1 function with following inputs:

$$x[n] = \left(\frac{1}{4}\right)^n u[n], a = [1 \quad -1.5 \quad 0.5], b = 1$$

Compare your results to verify that your filterdf2 function is correctly implemented.

```
fprintf('6(b)\n');
```

```
6(b)
```

```
n = 0:1:500;
x = (1/4).^n; a = [1 -1.5 0.5]; b = [1];
y = filterdf2(b,a,x)'
```

```
y = 1×501
    1.0000    1.7500    2.1875    2.4219    2.5430    2.6045    2.6355    2.6511 ⋯
```

```
y_verify = filterdf1(b,a,x)'
```

```
y_verify = 1×501
    1.0000    1.7500    2.1875    2.4219    2.5430    2.6045    2.6355    2.6511 ⋯
```

(Comment)

由上面的結果顯示，從 filterdf1 和 filterdf2 得到的結果相同。

## P7

The following numerator and denominator arrays in MATLAB represent the system function of a discrete-time system in direct form:

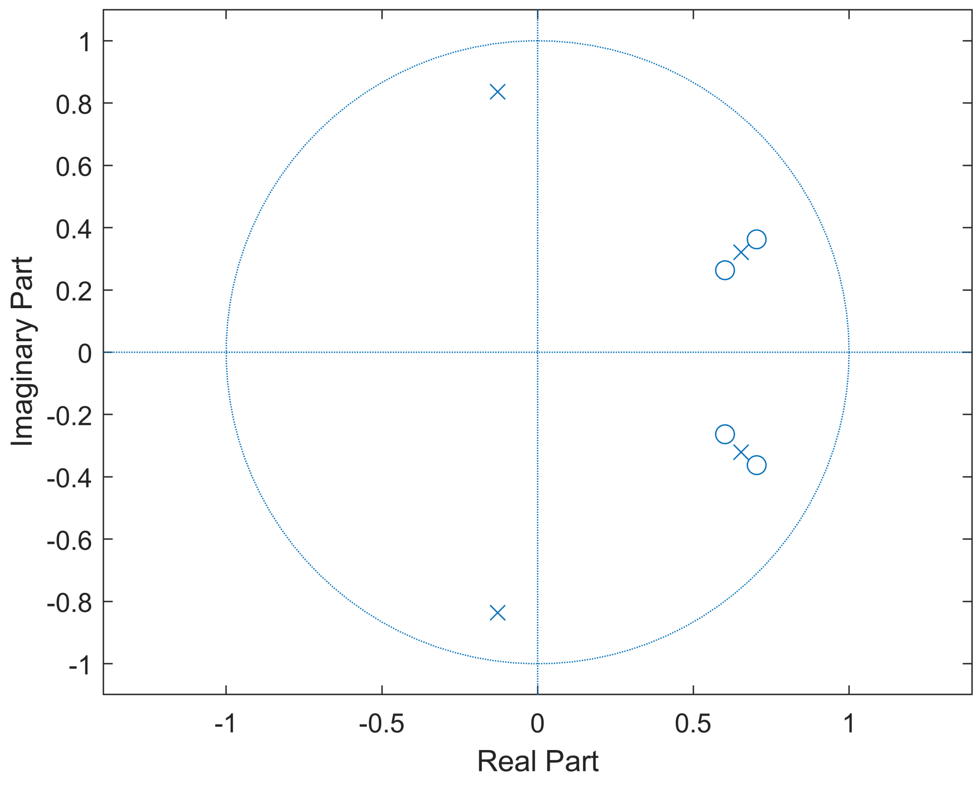
b = [1,-2.61,2.75,-1.36,0.27], a = [1,-1.05,0.91,-0.8,0.38].

Determine and draw each of the following structures:

```
close all; clear;
fprintf('7\n');
```

```
7
```

```
num = [1 -2.61 2.75 -1.36 0.27];
den = [1 -1.05 0.91 -0.8 0.38];
figure; zplane(num, den);
```



```
[num,den] = eqtflength(num,den);
[zero,pole,k] = tf2zp(num,den)
```

```
zero =
   0.7033
   0.7033
   0.6017
   0.6017
pole =
  -0.1288
  -0.1288
   0.6538
   0.6538
k = 1
```

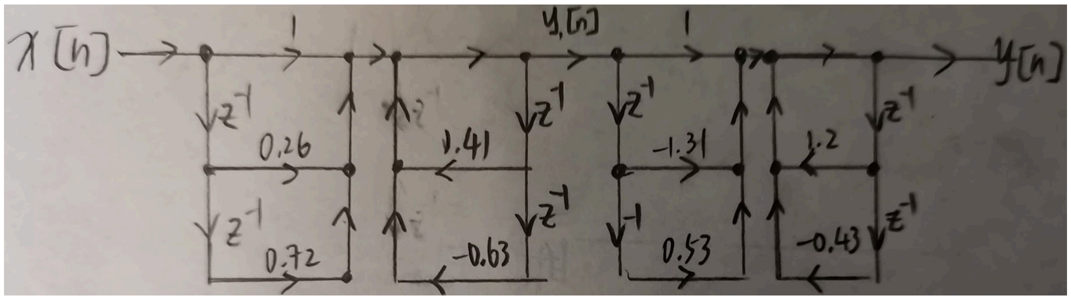
(Comment)

$$H(z)=\frac{1-2.61z^{-1}+2.75z^{-2}-1.36z^{-3}+0.27z^{-4}}{1-1.05z^{-1}+0.91z^{-2}-0.8z^{-3}+0.38z^{-4}}=\frac{(1+0.26z^{-1}+0.72z^{-2})(1-1.31z^{-1}+0.53z^{-2})}{(1-1.41z^{-1}+0.63z^{-2})(1-1.2z^{-1}+0.43z^{-2})}$$

(a) Cascade form with second-order sections in normal direct form I,

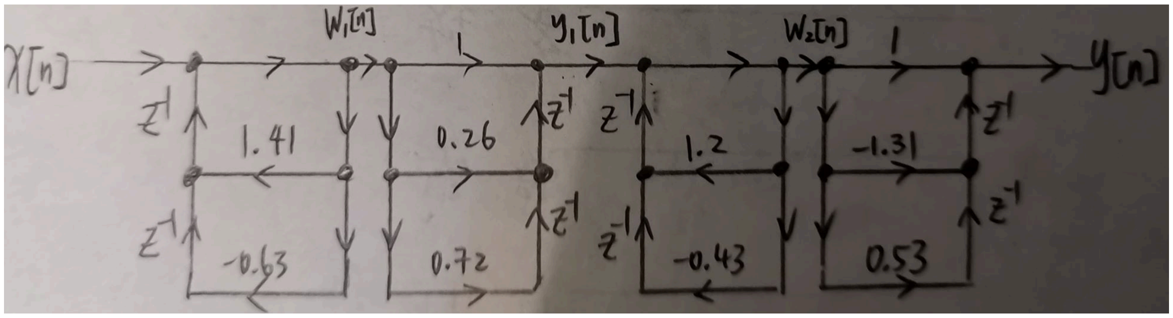
Ans.

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k], H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}$$



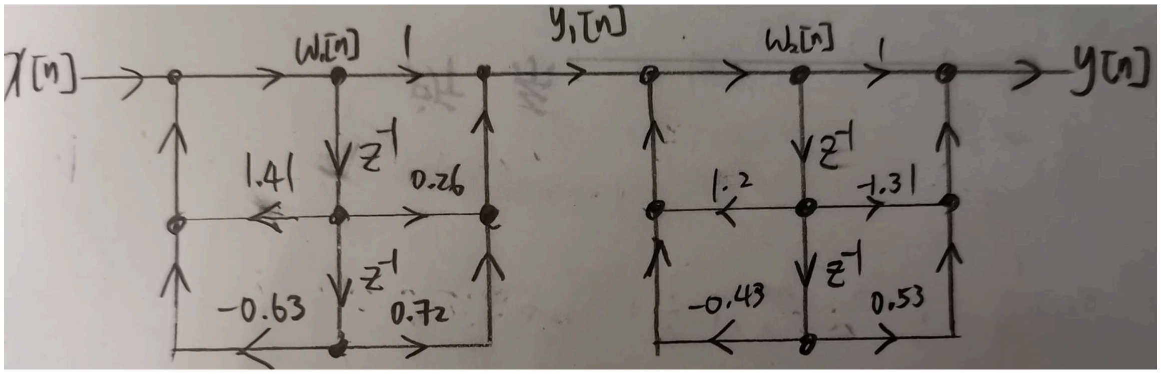
(b) Cascade form with second-order sections in transposed direct form I,

$$w[n] = -\sum_{k=1}^{N} a_k w[n-k] + x[n], y[n] = \sum_{k=0}^{M} b_k w[n-k]$$



(c) Cascade form with second-order sections in normal direct form II,

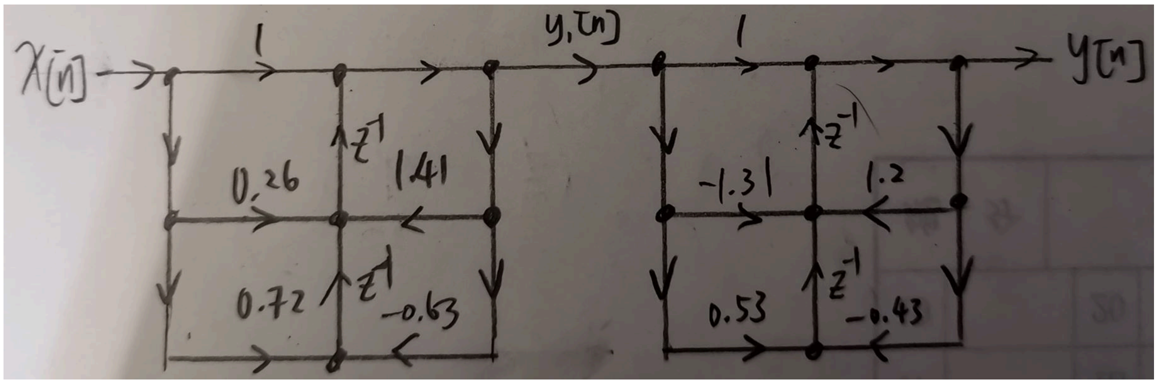
$$W(z) = \frac{1}{1 + \sum_{k=1}^{N} a_k z^{-k}}, Y(z) = \sum_{k=0}^{M} b_k z^{-k} W(z)$$



(d) Cascade form with second-order sections in transposed direct form II.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}},$$

$$Y(z) = z^{-1}V_1(z) + b_0 X(z), V_1(z) = z^{-1}V_2(z) - a_1 Y(z) + b_1 X(z), V_2(z) = b_2 X(z) - a_2 Y(z)$$



## P8

The frequency-sampling form is developed using (9.50) which uses complex arithmetic.

$$H(z) = \frac{1 - z^{-N}}{N} \sum_{k=0}^{N-1} \frac{H[k]}{1 - z^{-1} e^{\frac{j2\pi k}{N}}}, H[k] = H(z)\Big|_{z=e^{\frac{j2\pi k}{N}}} \quad (9.50)$$

(a) Develop a MATLAB function [G,sos]=firdf2fs(h) that determines frequency sampling form parameters given in (9.51) and (9.52) given the impulse response in h. The matrix sos (second order system) should contain second-order section coefficients in the form similar to the tf2sos function while G array should contain the respective gains of second-order sections. Incorporate the coefficients for the H[0] and H[N/2] terms in sos and G arrays.

$$H(z) = \frac{1 - z^{-N}}{N} \left\{ \frac{H[0]}{1 - z^{-1}} + \frac{H\left[\frac{N}{2}\right]}{1 + z^{-1}} + \sum_{k=1}^{K} 2|H[k]| H_k(z) \right\} \quad (9.51)$$

$$H_k(z) = \frac{\cos(\angle H[k]) - z^{-1}\cos\left(\angle H[k] - \frac{2\pi k}{N}\right)}{1 - 2\cos\left(\frac{2\pi k}{N}\right) z^{-1} + z^{-2}} \quad (9.52)$$

```
close all; clear;
fprintf('8(a)\n');
```

```
8(a)
```

```
open firdf2fs.m;
```

(b) Verify your function by input h with sampled frequency response (9.53) and compare with the system function (9.54) (see example 9.6 in the textbook)

$$H[k] = H\left(e^{\frac{j2\pi k}{33}}\right) = e^{\frac{-j2\pi k}{33}} \times \begin{cases} 1 & k = 0,1,2,31,32 \\ 0.5 & k = 3,30 \\ 0 & \text{otherwise} \end{cases} \quad (9.53)$$

$$H(z) = \frac{1-z^{-33}}{33}\left[\frac{1}{1-z^{-1}} + \frac{-1.99+1.99z^{-1}}{1-1.964z^{-1}+z^{-2}} + \frac{1.964-1.964z^{-1}}{1-1.857z^{-1}+z^{-2}} + \frac{-1.96+1.96z^{-1}}{1-1.683z^{-1}+z^{-2}}\right] \quad (9.54)$$

```
fprintf('8(b)\n');
```

```
8(b)
```

```
N = 33;
H = zeros(1, 33);
for k = 1:1:33
    if((k==1)||(k==2)||(k==3))
        H(k) = exp((-1*sqrt(-1)*2*pi*(k-1))/N);
    elseif((k==32)||(k==33))
        H(k) = exp((-1*sqrt(-1)*2*pi*(k-1))/N);
    elseif((k==4)||(k==31))
        H(k) = 0.5*(exp((-1*sqrt(-1)*2*pi*(k-1))/N));
    else
        H(k) = 0;
    end
end
h = ifft(H);
[G, sos] = firdf2fs(h);
sos_after = sos;
for a = 1:1:length(G)
    for b = 1:1:3
        sos_after(a,b) = sos_after(a,b)*G(a);
    end
end
sos_after % sos multiplied by the corresponding G factor
```

```
sos_after = 19×6 complex
   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i ···
   1.9639 + 0.0000i  -1.8567 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
   1.8567 + 0.0000i  -1.4475 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
   0.8413 + 0.0000i  -0.4154 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
  -0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
      ⋮
```

由此結果我們可以發現，經由 firdf2fs 處理後的值與 (9.54) 相近，一樣只有前面 4 項有值。

# HW6 Function by myself

**By: 105060012** 張育菘

```
function x = IDFT(X,N)
    x = fft(circfold(X,N))./N;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function W = dft_matrix(N)
    W = zeros(N, N);
    for a = 1:1:N
        for b = 1:1:N
            W(a,b) = exp(-1*sqrt(-1)*2*pi*(a-1)*(b-1)/N);
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function X = dftdirect_m(x,W)
    N = length(x);
    for a = 1:1:N
        sum = 0;
        for b = 1:1:N
            sum = sum + W(a,b)*x(b);
        end
        X(a) = sum;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function X = fftalt8(x)
    N = 8;
    W = exp(-1*sqrt(-1)*2*pi/N);
    for k = 1:1:4
        s1(k) = x(k) + x(k+4);
        s1(k+4) = x(k) - x(k+4);
    end
    for k = 1:1:2
        s2(k) = s1(k) + s1(k+2);
        s2(k+2) = s1(k+4) + ((W^2)*s1(k+6));
        s2(k+4) = s1(k) - s1(k+2);
        s2(k+6) = s1(k+4) - ((W^2)*s1(k+6));
    end
    s3(1) = s2(1) + s2(2);
    s3(2) = s2(3) + ((W^1)*s2(4));
    s3(3) = s2(5) + ((W^2)*s2(6));
    s3(4) = s2(7) + ((W^3)*s2(8));
    s3(5) = s2(1) - s2(2);
    s3(6) = s2(3) - ((W^1)*s2(4));
    s3(7) = s2(5) - ((W^2)*s2(6));
    s3(8) = s2(7) - ((W^3)*s2(8));

    X = s3;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function x = fftdifr2(x)
    % DIT Radix-2 FFT Algorithm
    N=length(x); nu=log2(N);
    for m = nu:-1:1 % stage
        L = 2^m;
        L2 = L/2;
        for ir = 1:L2
            W = exp(-1i*2*pi*(ir-1)/L);
            for it = ir:L:N
                ib = it + L2;
                temp = x(ib);
                temp_t = x(it);
                x(it) = temp_t + temp;
                x(ib) = W*(temp_t - temp);
            end
        end
    end
    x = bitrevorder(x); % Permute data into bit-reversed order
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function X = fftrecur_m(x,W)
    N = length(x);
    if N ==1
         X = x;
    else
         m = N/2;
         XE = fftrecur_m(x(1:2:N-1), W(1:m,1:2:N-1));
         XO = fftrecur_m(x(2:2:N), W(1:m,1:2:N-1));
         temp = W(1:m,2).*XO;
         X = [ XE+temp; XE-temp ];
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [y] = filterdf2(b,a,x)
    % Implementation of Direct Form II structure
    %  with zero initial conditions
    M = length(b)-1; N = length(a)-1; K = max(M,N);
    a0 = a(1); a = reshape(a,1,N+1)/a0;
    b = reshape(b,1,M+1)/a0; a = a(2:end);
    Lx = length(x); % x = [zeros(K,1);x(:)];
    Ly = Lx+K; y = zeros(Ly,1); w = zeros(Ly,1);
    for n = K+1:Ly
        w(n) = -a*w(n-1:-1:n-N) + x(n-K);
        y(n) = b*w(n:-1:n-M);
    end
    y = y(K+1:Ly);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [y, type] = filterfirlp(h,x)
    N = length(h); M = N - 1;
    h = reshape(h,1,M+1);
    x_len = length(x); x = [zeros(M,1);x(:)];
    y_len = x_len+N-1; y = zeros(1, y_len);
    if(mod(N,2)) % Type I, III
        h1 = h(1:1:((N-1)/2)); h2 = h(N:-1:((N+3)/2));
        if(h1 == h2) % Type I
            type = 'Type I';
            for n = M+1:1:y_len
                y(n) = h(1:1:M/2)*(x(n:-1:n-((M/2)-1))+x(n-M:1:n-M+((M/2)-1))) + h((M/2)+1)*x(
            end
        elseif((h1==-h2) & (h((N+1)/2)==0)) % Type III
            type = 'Type III';
            for n = M+1:1:y_len
                y(n) = h(1:1:M/2)*(x(n:-1:n-((M/2)-1))-x(n-M:1:n-M+((M/2)-1)));
            end
        else
            type = 'Not correspond to one of the four types.';
            for n = M+1:1:y_len
                y(n) = h(1:1:M+1)*x(n:-1:n-M);
            end
        end
    else % Type II, IV
        h1 = h(1:1:(N/2)); h2 = h(N:-1:((N+2)/2));
        if(h1 == h2) % Type II
            type = 'Type II';
            for n = M+1:1:y_len
                y(n) = h(1:1:(M+1)/2)*(x(n:-1:n-(((M+1)/2)-1))+x(n-M:1:n-M+(((M+1)/2)-1)));
            end
        elseif(h1 == -h2) % Type IV
            type = 'Type IV';
            for n = M+1:1:y_len
                y(n) = h(1:1:(M+1)/2)*(x(n:-1:n-(((M+1)/2)-1))-x(n-M:1:n-M+(((M+1)/2)-1)));
            end
        else
            type = 'Not correspond to one of the four types.';
            for n = M+1:1:y_len
                y(n) = h(1:1:M+1)*x(n:-1:n-M);
            end
        end
    end
    y = y(M+1:y_len);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [G,sos]=firdf2fs(h)
    H = fft(h); N = length(H);

    if(mod(N,2))
        K = (N+1)/2;
        N_even = 0;
    else
        K = (N/2);
        N_even = 1;
    end

    if(N_even)
        sos = [1 0 0 1 -1 0; zeros(K,6); 1 0 0 1 1 0];
        G = [H(1); zeros(K,1); H((N/2)+1)];
    else
        sos = [1 0 0 1 -1 0; zeros(K,6); 1 0 0 1 1 0];
        G = [H(1); zeros(K,1); 0];
    end

    for a = 2:1:K
        G(a) = 2*abs(H(a));
        thb1 = cos(angle(H(a)));
        thb2 = -cos(angle(H(a))-((2*pi*(a-1))/N));
        tha1 = -2*cos((2*pi*(a-1))/N);
        sos(a,:) = [thb1 thb2 0 1 tha1 1];
    end
end
```

# HW6 Function by reference

**By: 105060012** 張育菘

```
function Xdft=dftdirect(x)
% Direct computation of the DFT
N=length(x); Q=2*pi/N;
for k=1:N
     S=0;
     for n=1:N
         W(k,n)=exp(-j*Q*(k-1)*(n-1));
         S=S+W(k,n)*x(n);
     end
     Xdft(k)=S;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function x=fftditr2(x)
% DIT Radix-2 FFT Algorithm
N=length(x); nu=log2(N);
x = bitrevorder(x);
for m=1:nu
    L=2^m;
    L2=L/2;
    for ir=1:L2
        W=exp(-1i*2*pi*(ir-1)/L);
        for it=ir:L:N
            ib=it+L2;
            temp=x(ib)*W;
            temp_t=x(it);
            x(it)=temp_t+temp;
            x(ib)=temp_t-temp;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Xdft = fftrecur(x)
% Recursive computation of the DFT using divide & conquer
% N should be a power of 2
N = length(x);
if N ==1
  Xdft = x;
else
     m = N/2;
     XE = fftrecur(x(1:2:N));
     XO = fftrecur(x(2:2:N));
     W = exp(-2*pi*sqrt(-1)/N).^(0:m-1)';
     temp = W.*XO;
     Xdft = [ XE+temp ; XE-temp ];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [y] = filterdf1(b,a,x)
% Implementation of Direct Form I structure (Normal Form)
%  with zero initial conditions
% [y] = filterdf1(b,a,x)
M = length(b)-1; N = length(a)-1; K = max(M,N);
a0 = a(1); a = reshape(a,1,N+1)/a0;
b = reshape(b,1,M+1)/a0; a = a(2:end);
Lx = length(x); x = [zeros(K,1);x(:)];
Ly = Lx+K; y = zeros(Ly,1);
for n = K+1:Ly
    sn = b*x(n:-1:n-M);
    y(n) = sn - a*y(n-1:-1:n-N);
end
y = y(K+1:Ly);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [y] = filterfirdf(b,x)
% Implementation of FIR Direct Form structure (Normal Form)
% [y] = filterfirdf(b,a,x)
K = length(b)-1; b = reshape(b,1,K+1);
Lx = length(x); x = [zeros(K,1);x(:)];
Ly = Lx+K; y = zeros(1,Ly);
for n = K+1:Ly
    y(n) = b*x(n:-1:n-K);
end
y = y(K+1:Ly);
```