Experiment 5: Fast Fourier Transforms

1 Iterative Radix-2 DIF FFT

```
#include <stdio.h>
#include <math.h>
#include <complex.h>
#include <stdbool.h>

#define PI 3.1415926535897

static bool isPowerOfTwo(unsigned int n) {
    return n && ((n & (n - 1)) == 0);
}

void bitReverse(double complex x[], int N) {
    int bits = (int)log2(N);
    for (int i = 0; i < N; i++) {
        int rev = 0, temp = i;
        for (int b = 0; b < bits; b++) {
            rev = (rev << 1) | (temp & 1);
            temp >>= 1;
        }
        if (rev > i) {
            double complex tmp = x[i];
            x[i] = x[rev];
            x[rev] = tmp;
        }
    }
}

void fft(double complex x[], int N, int inverse) {
    int P = (int)log2(N);

    bitReverse(x, N);

    for (int s = 0; s < P; s++) {
        int m    = 1 << (s + 1);
        int half = m / 2;
        double angle = (inverse ? 2.0 : -2.0) * PI / m;
        double complex Wm = cos(angle) + I * sin(angle);

        for (int b = 0; b < N; b += m) {
            double complex w = 1.0;
            for (int j = 0; j < half; j++) {
                double complex u = x[b + j];
                double complex t = w * x[b + j + half];
                x[b + j]        = u + t;
```

```
                x[b + j + half] = u - t;
                w *= Wm;
            }
        }

        printf("\nStage %d output:\n", s + 1);
        for (int i = 0; i < N; i++) {
            printf("x[%d] = %.2f %+.2fi\n", i, creal(x[i]), cimag(x[i]));
        }
    }

    // IFFT scaling
    if (inverse) {
        for (int i = 0; i < N; i++) x[i] /= N;
    }
}

int main() {
    unsigned int N;
    printf("Enter value for N: ");
    scanf("%u", &N);

    if (!isPowerOfTwo(N)) {
        printf("Error: N must be a power of 2!\n");
        return 1;
    }

    double complex x[N];
    double real, imag;

    for (int i = 0; i < N; i++) {
        printf("Enter real part of x[%d]: ", i);
        scanf("%lf", &real);
        printf("Enter imag part of x[%d]: ", i);
        scanf("%lf", &imag);
        x[i] = real + imag * I;
    }

    printf("\nInput:\n");
    for (int i = 0; i < N; i++) {
        printf("x[%d] = %.2f %+.2fi\n", i, creal(x[i]), cimag(x[i]));
    }

    // FFT
    fft(x, N, 0);
    printf("\nFFT output:\n");
    for (int i = 0; i < N; i++) {
        printf("X[%d] = %.2f %+.2fi\n", i, creal(x[i]), cimag(x[i]));
```

```
    }

    fft(x, N, 1);
    printf("\nIFFT output:\n");
    for (int i = 0; i < N; i++) {
        printf("x[%d] = %.2f %+.2fi\n", i, creal(x[i]), cimag(x[i]));
    }

    return 0;
}
```

DIT OUTPUT

```
Enter value for N: 8
Enter real part of x[0]: 1
Enter imag part of x[0]: 1
Enter real part of x[1]: 2
Enter imag part of x[1]: -1
Enter real part of x[2]: -1
Enter imag part of x[2]: 2
Enter real part of x[3]: 0
Enter imag part of x[3]: -3
Enter real part of x[4]: 3
Enter imag part of x[4]: 0
Enter real part of x[5]: -2
Enter imag part of x[5]: 1
Enter real part of x[6]: 1
Enter imag part of x[6]: -2
Enter real part of x[7]: -1
Enter imag part of x[7]: 0

Input:
x[0] = 1.00 +1.00i
x[1] = 2.00 -1.00i
x[2] = -1.00 +2.00i
x[3] = 0.00 -3.00i
x[4] = 3.00 +0.00i
x[5] = -2.00 +1.00i
x[6] = 1.00 -2.00i
x[7] = -1.00 +0.00i

Stage 1 output:
x[0] = 4.00 +1.00i
x[1] = -2.00 +1.00i
x[2] = 0.00 +0.00i
x[3] = -2.00 +4.00i
x[4] = 0.00 +0.00i
x[5] = 4.00 -2.00i
x[6] = -1.00 -3.00i
x[7] = 1.00 -3.00i

Stage 2 output:
x[0] = 4.00 +1.00i
x[1] = 2.00 +3.00i
x[2] = 4.00 +1.00i
x[3] = -6.00 -1.00i
x[4] = -1.00 -3.00i
x[5] = 1.00 -3.00i
x[6] = 1.00 +3.00i
```

```
x[7] = 7.00 -1.00i

Stage 3 output:
x[0] = 3.00 -2.00i
x[1] = 0.59 +0.17i
x[2] = 7.00 +0.00i
x[3] = -11.66 -5.24i
x[4] = 5.00 +4.00i
x[5] = 3.41 +5.83i
x[6] = 1.00 +2.00i
x[7] = -0.34 +3.24i

FFT output:
X[0] = 3.00 -2.00i
X[1] = 0.59 +0.17i
X[2] = 7.00 +0.00i
X[3] = -11.66 -5.24i
X[4] = 5.00 +4.00i
X[5] = 3.41 +5.83i
X[6] = 1.00 +2.00i
X[7] = -0.34 +3.24i

Stage 1 output:
x[0] = 8.00 +2.00i
x[1] = -2.00 -6.00i
x[2] = 8.00 +2.00i
x[3] = 6.00 -2.00i
x[4] = 4.00 +6.00i
x[5] = -2.83 -5.66i
x[6] = -12.00 -2.00i
x[7] = -11.31 -8.49i

Stage 2 output:
x[0] = 16.00 +4.00i
x[1] = 0.00 -0.00i
x[2] = 0.00 +0.00i
x[3] = -4.00 -12.00i
x[4] = -8.00 +4.00i
x[5] = 5.66 -16.97i
x[6] = 16.00 +8.00i
x[7] = -11.31 +5.66i

Stage 3 output:
x[0] = 8.00 +8.00i
x[1] = 16.00 -8.00i
x[2] = -8.00 +16.00i
x[3] = 0.00 -24.00i
x[4] = 24.00 +0.00i
```

```
x[5] = -16.00 +8.00i
x[6] = 8.00 -16.00i
x[7] = -8.00 +0.00i

IFFT output:
x[0] = 1.00 +1.00i
x[1] = 2.00 -1.00i
x[2] = -1.00 +2.00i
x[3] = 0.00 -3.00i
x[4] = 3.00 +0.00i
x[5] = -2.00 +1.00i
x[6] = 1.00 -2.00i
x[7] = -1.00 +0.00i
```

1 Iterative Radix-2 DIF FFT

```
#include <stdio.h>
#include <math.h>
#include <complex.h>
#include <stdbool.h>

#define PI 3.1415926535897

static bool isPowerOfTwo(unsigned int n) {
    return n && ((n & (n - 1)) == 0);
}

void bitReverse(double complex x[], int N) {
    int bits = log2(N);
    for (int i = 0; i < N; i++) {
        int rev = 0, temp = i;
        for (int b = 0; b < bits; b++) {
            rev = (rev << 1) | (temp & 1);
            temp >>= 1;
        }
        if (rev > i) {
            double complex tmp = x[i];
            x[i] = x[rev];
            x[rev] = tmp;
        }
    }
}
void fft(double complex x[], int N, int inverse) {
    int P = log2(N);

    for (int s = 0; s < P; s++) {
        int m = 1 << (P-s);
        int half = m / 2;
        double angle = (inverse ? 2.0 : -2.0) * PI / m;
        double complex Wm = cos(angle) + I * sin(angle);

        for (int b = 0; b < N; b += m) {
            double complex w = 1.0;
            for (int j = 0; j < half; j++) {
                double complex u = x[b + j];
                double complex v = x[b + j + half];
                x[b + j] = u + v;
                x[b + j + half] = (u - v) * w;
                w *= Wm;   // twiddle factor update
            }
        }
        printf("\nStage %d output:\n", s + 1);
```

```
        for (int i = 0; i < N; i++) {
            printf("x[%d] = %.2f %+.2fi\n", i, creal(x[i]), cimag(x[i]));
        }
    }
    bitReverse(x, N);

    if (inverse) {
        for (int i = 0; i < N; i++) {
            x[i] /= N;
        }
    }
}

int main() {
    unsigned int N;
    printf("Enter value for N: ");
    scanf("%u", &N);

    if (!isPowerOfTwo(N)) {
        printf("Error: N must be a power of 2!\n");
        return 1;
    }

    double complex x[N];
    double real, imag;

    for (int i = 0; i < N; i++) {
        printf("Enter real part of x[%d]: ", i);
        scanf("%lf", &real);
        printf("Enter imag part of x[%d]: ", i);
        scanf("%lf", &imag);
        x[i] = real + imag * I;
    }

    printf("\nInput :\n");
    for (int i = 0; i < N; i++) {
        printf("x[%d] = %.2f %+.2fi\n", i, creal(x[i]), cimag(x[i]));
    }

    fft(x, N, 0);
    printf("\nFFT output:\n");
    for (int i = 0; i < N; i++) {
        printf("X[%d] = %.2f %+.2fi\n", i, creal(x[i]), cimag(x[i]));
    }


    fft(x, N, 1);
    printf("\nIFFT output:\n");
```

```
    for (int i = 0; i < N; i++) {
        printf("x[%d] = %.2f %+.2fi\n", i, creal(x[i]), cimag(x[i]));
    }
    return 0;
}
```

DIF OUTPUT

```
Enter value for N: 8
Enter real part of x[0]: 2
Enter imag part of x[0]: 1
Enter real part of x[1]: -1
Enter imag part of x[1]: 2
Enter real part of x[2]: 0
Enter imag part of x[2]: -1
Enter real part of x[3]: 3
Enter imag part of x[3]: 3
Enter real part of x[4]: -2
Enter imag part of x[4]: 0
Enter real part of x[5]: 1
Enter imag part of x[5]: -1
Enter real part of x[6]: -3
Enter imag part of x[6]: 2
Enter real part of x[7]: 4
Enter imag part of x[7]: 0

Input :
x[0] = 2.00 +1.00i
x[1] = -1.00 +2.00i
x[2] = 0.00 -1.00i
x[3] = 3.00 +3.00i
x[4] = -2.00 +0.00i
x[5] = 1.00 -1.00i
x[6] = -3.00 +2.00i
x[7] = 4.00 +0.00i

Stage 1 output:
x[0] = 0.00 +1.00i
x[1] = 0.00 +1.00i
x[2] = -3.00 +1.00i
x[3] = 7.00 +3.00i
x[4] = 4.00 +1.00i
x[5] = 0.71 +3.54i
x[6] = -3.00 -3.00i
x[7] = 2.83 -1.41i
```

```
Stage 2 output:
x[0] = -3.00 +2.00i
x[1] = 7.00 +4.00i
x[2] = 3.00 +0.00i
x[3] = -2.00 +7.00i
x[4] = 1.00 -2.00i
x[5] = 3.54 +2.12i
x[6] = 7.00 +4.00i
x[7] = 4.95 +2.12i

Stage 3 output:
x[0] = 4.00 +6.00i
x[1] = -10.00 -2.00i
x[2] = 1.00 +7.00i
x[3] = 5.00 -7.00i
x[4] = 4.54 +0.12i
x[5] = -2.54 -4.12i
x[6] = 11.95 +6.12i
x[7] = 2.05 +1.88i

FFT output:
X[0] = 4.00 +6.00i
X[1] = 4.54 +0.12i
X[2] = 1.00 +7.00i
X[3] = 11.95 +6.12i
X[4] = -10.00 -2.00i
X[5] = -2.54 -4.12i
X[6] = 5.00 -7.00i
X[7] = 2.05 +1.88i

Stage 1 output:
x[0] = -6.00 +4.00i
x[1] = 2.00 -4.00i
x[2] = 6.00 +0.00i
x[3] = 14.00 +8.00i
x[4] = 14.00 +8.00i
x[5] = 2.00 +8.00i
x[6] = -14.00 -4.00i
x[7] = -10.00 +4.00i

Stage 2 output:
x[0] = 0.00 +4.00i
x[1] = 16.00 +4.00i
x[2] = -12.00 +4.00i
x[3] = 12.00 -12.00i
x[4] = 0.00 +4.00i
x[5] = -8.00 +12.00i
```

```
x[6] = 28.00 +12.00i
x[7] = -4.00 +12.00i

Stage 3 output:
x[0] = 16.00 +8.00i
x[1] = -16.00 +0.00i
x[2] = 0.00 -8.00i
x[3] = -24.00 +16.00i
x[4] = -8.00 +16.00i
x[5] = 8.00 -8.00i
x[6] = 24.00 +24.00i
x[7] = 32.00 +0.00i

IFFT output:
x[0] = 2.00 +1.00i
x[1] = -1.00 +2.00i
x[2] = 0.00 -1.00i
x[3] = 3.00 +3.00i
x[4] = -2.00 +0.00i
x[5] = 1.00 -1.00i
x[6] = -3.00 +2.00i
x[7] = 4.00 +0.00i
```