Implementarea transformatei Fourier

Se bazeaza pe o reprezentare matriciala. Consideram $N_1 = N_2$ si notam cu $Z = \exp(-j\frac{2\pi}{N})$

Fie matricea Z:

$$Z = \begin{vmatrix} 1 & 1 & 1 & \_ & \_ & \_ & \_ & \_ & 1 \\ 1 & z & z^2 & \_ & \_ & \_ & \_ & \_ & z^{N-1} \\ 1 & z^2 & z^4 & \_ & \_ & \_ & \_ & \_ & z^{2(N-1)} \\ \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\ 1 & z^{N-1} & z^{2(N-1)} & \_ & \_ & \_ & \_ & \_ & z^{(N-1)(N-1)} \end{vmatrix}$$

Z are N*N elemente. $Z[i][j] = z^{i*j}$, $0 \le i, j \le N-1$

In majoritatea cazurilor, N este o putere a lui Z. Matricea Z poate fi simplificata observand ca:

$$Z^N = 1, Z^{N/2} = -1, Z^{N/4} = -j, Z^{3N/4} = j$$

Alte ***proprietati ale matricei Z:***

1. *Este simetrica*
2. *Produsul scalar al oricaror 2 randuri sau 2 coloane este egal cu zero*
3. *Produsul scalar al unei coloane cu ea insasi sau al unui rand cu el insusi este egal cu N*
4. *Inversa sa este egala cu conjugata transpusei *1/N.*

Transformata Fourier 1D se poate rescrie astfel:

$$F(u)=\sum_{x=0}^{N-1} f(x)\exp(-j\frac{2\pi}{N}ux)=\sum_{x=0}^{N-1} f(x)*z^{u*x}=\sum_{x=0}^{N-1} f(x)*Z[u][x]$$

Rezulta *forma matriciala:*

F=Z*f

iar pentru *transformata Fourier bidimensionala:*

F=Z*f* Z^T, unde

F este *matricea imagine iar F- matricea transformatei Fourier a imaginii.*

Aplicarea directa a formulei matriciale conduce la N^4 inmultiri cu numere complexe si $N^2(N^2-1)$ adunari. De aceea, au fost eleborati algoritmi pentru calculul transformatei Fourier. Astfel este *algoritmul FFT (Fast Fourier Transform)* prezentat in continuare.

Transformata Fourier rapida 2D foloseste transformata Fourier rapida 1D. Mai exact, transformata Fourier 2D este implementata ca o transformare in 2 pasi.

Valorile din matricile F si Z sunt numere complexe:

$$z = e^{(-j\frac{2\pi}{N})} = \cos(-\frac{2\pi}{N}) + j * \sin(-\frac{2\pi}{N})$$

Transformata Fourier rapida 1D

$$F(u) = \sum_{k=0}^{N-1} f(k) * z^{uk}, 0 \le u \le N-1$$

Se inlocuieste k cu m, astfel incat $0 \le m \le \frac{N}{2} - 1$:

K = 2m pentru k par;

K = 2m+1 pentru k impar

Rescriem formula transformarii, punand in evidenta termenii pari si cei impari:

$$F(u) = \sum_{m=0}^{N/2-1} [f(2m) * Z^{u*2m} + f(2m+1) * z^{u*(2m+1)}]$$

Notam : M=N/2;

g= $e^{(-j\frac{2\pi}{M})}$, $g = z^2$

Atunci:

$$F(u) = \sum_{m=0}^{M-1} f_p(m) * g^{u*m} + z^u \sum_{m=0}^{M-1} f_i(m) * g^{u*m}, 0 \le u \le N-1$$

$$f_p(m) = f(2m)$$

$$f_i(m) = f(2m+1)$$

sau

$$F(u) = F_p(u) + e^{(-j\frac{2\pi u}{N})} * F_i(u)$$

Pentru N=2 (M=1): $F(0) = f(0)*g^0 + z^0*f(1) = f(0) + f(1)$

$F(1) = f(0) - f(1)$

Implementarea algoritmului in C:

```
void FFT_1D(int N, complex *f)
{ // intoarce in f transformata Fourier
        int n;
        complex *fp, *fi, x;
          if(N==2)
                { x=f[0];
```

```
            f[0] = f[0] + f[1];
            f[1] = x - f[1];
            return;
           }
    fp = (complex *) malloc (N/2 * sizeof (complex));
    fi = (complex *) malloc  (N/2 * sizeof (complex));
    for(m=0 ; m<N/2; m++)
        {       fp[m] = f[2*m];
                fi[m] = f[2*m+1];
        }
    FFT_1D (N/2, fp);
    FFT_1D (N/2, fi);
    for (i=0; i<N/2; i++)
        {       r   = cos (-2*M_Pi/N*i);
                im = sin (-2*M_Pi/N*i);
                f[i].re = fp[i].re + r*fi[i].re;
                f[i].im = fp[i].im + im*fi[i].im;
        }
    free (fp);
   free (fi);
}
```

Transformata Fourier rapida 2D

Este realizata ca o transformare in 2 pasi, folosind transformarea Fourier rapida 1 D:

1. *Se determina transformata Fourier a fiecarei coloane a matricei imagine;*
2. *Se determina transformata Fourier a fiecarui rand din matricea obtinuta in prima etapa.*

Implementarea algoritmului in C:

```
void FFT_2D (int N, complex **f)
{ complex *x = (complex* ) malloc (N * sizeof (complex));
  for (int k=0; k<N; k++)
       {  for (int l=0; l<N; l++) //copiaza o coloana in x;
              x[l] = f[l][k]
          FFT_1D(N,x);
          for(l=0; l<N; l++)
              f[l][k] = x[l];
```

```
        }

  for (l=0; l<N; l++) // pentru fiecare rand
        {  for (k=0; k<N; k++)
                x[k] = f[l][k];
           FFT_1D(N, x);
           for (k=0; k<N; k++)
                f[l][k] = x[k];
        }
}
```

Afisarea transformatei Fourier a unei imagini

In unele operatii de prelucrare a imaginilor (restaurare, filtrare) se lucreaza cu marimea si faza transformatei Fourier a imaginii.

Fie **F** *matricea in care s-a memorat transformata Fourier a imaginii*, **FM** – *matricea in care s-a memorat marimea semnalului* si **FF** *matricea in care se memoreaza faza*:

$$FM[k_1][k_2] = \sqrt{F[k_1][k_2].re^2 + F[k_1][k_2].im^2}, 0 \le k_1, k_2 \le N-1$$

$$FF[k1][k2] = \begin{cases} \text{atan2}(F[k1][k2].im \,/\, F[k1][k2].re) & \text{pentru } F[k1][k2].re > 0 \\ \pi/2 & \text{pentru } F[k1][k2].re = 0 \text{ si } F[k1][k2].im >= 0 \\ -\pi/2 & \text{pentru } F[k1][k2].re = 0 \text{ si } F[k1][k2].im < 0 \end{cases}$$

Prin analiza marimii transformatei Fourier se pot obtine informatii utile despre frecventa semnalului 2D reprezentand imaginea. In acest scop, se transforma marimea transformatei Fourier intr-o imagine. Elementele matricei FM sunt scalate astfel inact sa poata fi afisate prin 256 nivele de gri. Totodata, asupra imaginii astfel obtinute se efectueaza o *translatie circulara*, prin care punctul corespunzator elementului F[0][0] (originea in spatiul Fourier) este adus in centrul ecranului. Se efectueaza o scalare la scara logaritmica pentru a fi posibila afisarea unui domeniu relativ mare de valori.

Translatia circulara a unei secvente discrete 2D, f(x,y), este definita astfel:

$$f'(x,y) = f((x+m1) \bmod N1, (y+m2) \bmod N2), \qquad (x,y) \in R_{N1,N2}$$

Pentru N1 = N2 = N, m1 = m2 = N/2, translatia circulara este:

1	2
3	4

4	3
2	1

Programul prezentat in continuare realizeaza *transformarea matricei marime intr-o matrice imagine cu 256 nivele de gri*:

```
#define N 1024
typedef unsigned char ** imagine;
complex F[N][N];
float FM[N][N], t, max;
imagine img;
int r,c;
max=0;
//calculam marimea transformatei (amplitudinea) si  valoarea maxima din FM
for (r=0; r<N; r++)
        for (c=0; c<N; c++)
                { t = sqrt (F[r][c].re * F[r][c].re + F[r][c].im * F[r][c].im );
                  FM[r][c] = t;
```

```
            if(t>max)max = t;
        }
//calculam imaginea si efectuam translatii circulare
    coef = 255.0/log(1.0 + max);
    for( r = 0; r< N/2; r++)
        { for(c=0; c<N/2; c++)
            img[r][c] = (unsigned char )(0.5 + log(1.0 + FM[r + N/2][c + N/2])*coef);
          for(c=N/2; c<N; c++)
            img[r][c] = (unsigned char )(0.5 + log(1.0 + FM[r + N/2][c - N/2])*coef);
        }
    for( r = N/2; r< N; r++)
        { for(c=0; c<N/2; c++)
            img[r][c] = (unsigned char )(0.5 + log(1.0 + FM[r - N/2][c + N/2])*coef);
          for(c=N/2; c<N; c++)
            img[r][c] = (unsigned char )(0.5 + log(1.0 + FM[r - N/2][c - N/2])*coef);
        }
```