MAC0317/MAC5920

Introdução ao Processamento de Sinais Digitais

Seção 2.6: Transformada Rápida de Fourier (FFT)

- A FFT é uma maneira eficiente de implementar a DFT, explorando a estrutura da matriz F;
- Forma atual se deve a Cooley & Tuckey (1966), mas partes da construção já existiam;
- A DFT implementada de forma ingênua tem complexidade $\mathcal{O}(N^2)$, pois cada coeficiente $X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}$ requer $2N-1$ operações (N multiplicações e $N-1$ somas), portanto para calcular $X = DFT(x) \in \mathbb{C}^N$ são necessárias $2N^2 - N$ operações.
- É possível explorar as simetrias da DFT quando x é real, como por exemplo $X_{N-k} = \overline{X_k},\ k = 1, \dots, \frac{N}{2} - 1$, mas o custo continua proporcional a N^2.

A ideia simples do cálculo da DFT por recursão reside em considerar $N = 2^B$ e particionar o vetor $x = (x_0, x_1, \dots, x_{N-1})$ em duas partes:

$$x_{\text{par}} = (x_0, x_2, \dots, x_{N-2})$$

e

$$x_{\text{ímpar}} = (x_1, x_3, \dots, x_{N-1}),$$

e calcular separadamente as DFTs de cada um destes vetores, combinando os resultados. Esta estratégia é conhecida como *Divisão e Conquista*.

Note que

$$
\begin{aligned}
X_k &= \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}} \\
&= \sum_{n \text{ par}} x_n e^{-i2\pi k \frac{n}{N}} + \sum_{n \text{ ímpar}} x_n e^{-i2\pi k \frac{n}{N}} \\
&= \sum_{n=0}^{N/2-1} x_{2n} e^{-i2\pi k \frac{2n}{N}} + \sum_{n=0}^{N/2-1} x_{2n+1} e^{-i2\pi k \frac{2n+1}{N}} \\
&= \sum_{n=0}^{N/2-1} (x_{\text{par}})_n e^{-i2\pi k \frac{n}{N/2}} + e^{-i2\pi k \frac{1}{N}} \sum_{n=0}^{N/2-1} (x_{\text{ímpar}})_n e^{-i2\pi k \frac{n}{N/2}} \\
&= (X_{\text{par}})_k + e^{-i2\pi k \frac{1}{N}} (X_{\text{ímpar}})_k
\end{aligned}
$$

Na expressão

$$\begin{aligned} DFT(x)_k &= X_k = (X_{\text{par}})_k + e^{-i2\pi k\frac{1}{N}}(X_{\text{ímpar}})_k \\ &= DFT(X_{\text{par}})_k + e^{-i2\pi k\frac{1}{N}} DFT(x_{\text{ímpar}})_k, \end{aligned}$$

as duas últimas DFTs são periódicas com período $\frac{N}{2}$, e portanto para calcular todos os valores X_k com $k = 0, 1, \dots, N-1$, basta obter

$$\left.\begin{array}{l} \text{DFT}(x_{\text{par}})_k \\ \text{DFT}(x_{\text{ímpar}})_k \end{array}\right\} \text{ para } k = 0, 1, \dots, \frac{N}{2} - 1.$$

Essa é a forma recursiva da implementação da FFT.

FFT(x)

base: se $x \in \mathbb{C}^1$, *devolva* $X = x$

calcule $X_{par} = FFT([x_0, x_2, \ldots, x_{N-2}])$

calcule $X_{ímpar} = FFT([x_1, x_3, \ldots, x_{N-1}])$

para $k = 0, \ldots, N-1$ *faça*

$$X_k = (X_{par})_k + e^{-i2\pi k \frac{1}{N}} (X_{ímpar})_k$$

devolva X

Sendo $C(N/2)$ o custo de computar cada FFT dos subvetores X_{par} e $X_{\text{ímpar}}$, e levando em consideração que o custo de combinar as soluções é proporcional a N (uma multiplicação e uma soma para cada coeficiente X_k), o custo da chamada FFT(x) será

$$
\begin{aligned}
C(N) &= 2C(N/2) + \alpha N \qquad (\alpha \text{ constante}) \\
&= 2\,(2C(N/4) + \alpha N/2) + \alpha N = 4C(N/4) + 2\alpha N \\
&= 4\,(2C(N/8) + \alpha N/4) + 2\alpha N = 8C(N/8) + 3\alpha N = 2^3 C(N/2^3) + 3\alpha N \\
&= 2^4 C(N/2^4) + 4\alpha N \\
&= \quad \cdots \quad (\text{depois de } B \text{ passos}) \quad \cdots \\
&= 2^B C(N/2^B) + B\alpha N \\
&= NC(1) + \alpha N B \\
&= \beta N + \alpha N \log N \qquad (\beta \text{ constante}) \\
&= \mathcal{O}(N \log N)
\end{aligned}
$$

Implementação em Python da DFT ingênua

In [3]:

```
def DFTingenua(x):
    N = len(x); n = np.array(range(N))
    X = np.ndarray(N, dtype=complex)
    for k in range(N):
        X[k] = sum(x*np.exp(-1j*2*m.pi*k*n/N))
    return X

# teste simples: DFT(Dirac)=[1,1,...,1]
x = [ 1, 0, 0, 0 ]
X = DFTingenua(x)
print(X)
```

```
[1.+0.j 1.+0.j 1.+0.j 1.+0.j]
```

Implementação em Python da FFT

In [63]:

```
def FFT(x):
    N = len(x);
    if N<=1: return x
    X = np.ndarray(N, dtype=complex)
    Xpar = FFT(x[0:N:2])
    Ximpar = FFT(x[1:N:2])
    for k in range(N):
        X[k] = Xpar[k%(N//2)]+np.exp(-1j*2*m.pi*k/N)*Ximpar[k%(N//2)]
    return X

# teste simples: DFT(Dirac)=[1,1,...,1]
x = [ 1, 0, 0, 0 ]
X = FFT(x)
print(X)
```

```
[1.+0.j 1.+0.j 1.+0.j 1.+0.j]
```

Comparação de tempos entre DFT e FFT

In [64]:

```
# comparação de tempo
N = np.arange(50,2000,50) # tamanhos de vetor
TDFT = np.zeros(len(N)) # tempos de execução
TFFT = np.zeros(len(N)) # tempos de execução
R = 10 # número de repetições para cada N
print("Rodando experimento...")
for r in range(R):
    for i in range(len(N)):
        x = np.random.rand(N[i])
        t = time(); DFTingenua(x); TDFT[i] += (time()-t)/R
        t = time(); FFT(x); TFFT[i] += (time()-t)/R
    print("{}% completado...".format(100*(r+1)/R))
print("Pronto!")
```

```
Rodando experimento...
10.0% completado...
20.0% completado...
30.0% completado...
40.0% completado...
50.0% completado...
60.0% completado...
70.0% completado...
80.0% completado...
90.0% completado...
100.0% completado...
Pronto!
```

In [65]:

```
plt.plot(N,TDFT,label="DFT")
plt.plot(N,TFFT,label="FFT")
plt.legend()
plt.show()
```