FFT a její aplikace

FFT a její aplikace

David Bernhauer, 2014

```
import cmath

def printComplexNumber ( x ):
    print '%.2f + %.2f j' % ( x.real, x.imag )

def printListOfComplexNumber ( l ):
    for i in range( len( l ) ):
        printComplexNumber( l[ i ] )
```

Diskrétní Fourierova transformace

Diskrétní Fourierova tranformace převádí původní diskrétní signál do frekvenčního pásma. Je definována tímto vztahem:

$$X_k = \sum_{n = 0}^{N-1} x_n \cdot e^{-i \frac{2 \pi}{N} n k}, \text{ kde } k = 0, 1, \dots, N - 1$$

```
def Xk ( original, N, k ):
    sum = 0
    for n in range( N ):
        sum += original[ n ] * cmath.exp( -1j * 2 * cmath.pi * k * n / N )
    return sum

def dft ( original ):
    N = len( original )
    frequency = []
    for k in range( N ):
        frequency.append( Xk( original, N, k ) )
    return frequency

printListOfComplexNumber( dft( [9,7,9,2] ) )
```

```
27.00 + 0.00 j
-0.00 + -5.00 j
9.00 + 0.00 j
-0.00 + 5.00 j
```

Inverzní transformace k DFT je definována velice podobně tímto vztahem:

$$x_n = \frac{1}{N-1} \sum_{k=0}^{N} X_k \cdot e^{i \frac{2\pi}{N} nk},$$
kde $n = 0, 1, ..., N - 1$

Důkaz: Abychom si ověřili tyto dva vztahy stačí DFT dosadit do vzorce pro inverzní DFT.

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m = 0}^{N-1} x_m \cdot e^{-i \frac{2\pi}{N} mk} \cdot e^{i \frac{2\pi}{N} nk} =\\= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m = 0}^{N-1} x_m \cdot e^{-i \frac{2\pi}{N} mk + i \frac{2\pi}{N} nk} =\\= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m = 0}^{N-1} x_m \cdot e^{i \frac{2\pi}{N} k \left(n - m \right)} =\\= \frac{1}{N} \sum_{m = 0}^{N-1} x_m \sum_{k=0}^{N-1} \cdot e^{i \frac{2\pi}{N} k \left(n - m \right)} =\\= \frac{1}{N} \left(\sum_{m = 0}^{n-1} x_m \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k \left(n - m \right)} + x_n \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k 0} + \sum_{m = n+1}^{N-1} x_m \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k \left(n - m \right)} \right) =\\= \frac{1}{N} \left(\sum_{m = 0}^{n-1} x_m \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k \left(n - m \right)} + x_n \sum_{k=0}^{N-1} 1 + \sum_{m = n+1}^{N-1} x_m \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k \left(n - m \right)} \right) =\\= x_n + \frac{1}{N} \left(\sum_{m = 0}^{n-1} x_m \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k \left(n - m \right)} + \sum_{m = n+1}^{N-1} x_m \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k \left(n - m \right)} \right)$$

Z této rovnosti by tedy mělo platit, že $\frac{1}{N} \left(\sum_{m = 0}^{n-1} x_m \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k \left(n - m \right)} + \sum_{m = n+1}^{N-1} x_m \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k \left(n - m \right)} \right) = 0$

Jelikož jsou proměnné x_i nezávislé, pak nutně musí platit, že $\forall m \neq n$ platí $x_m \sum_{k=0}^{N-1} e^{i \frac{2\pi}{N} k \left(n - m \right)} = 0$.

Připomeňme si vztah $e^{i\theta} = \cos\left(\theta\right) + i\sin\left(\theta \right)$. Můžeme pak psát:

$$\sum_{k=0}^{N-1} \left(\cos\left(\frac{2\pi}{N} k \left(n - m \right) \right) + i\sin\left(\frac{2\pi}{N} k \left(n - m \right) \right) \right) = 0 \\ \sum_{k=0}^{N-1} \cos\left(\frac{2\pi}{N} k \left(n - m \right) \right) + i \sum_{k=0}^{N-1} \sin\left(\frac{2\pi}{N} k \left(n - m \right) \right) = 0 \\ \sum_{k=0}^{N-1} \cos\left(k \frac{2\pi\left(n - m \right)}{N} \right) = 0 \wedge \sum_{k=0}^{N-1} \sin\left(k \frac{2\pi\left(n - m \right)}{N} \right) = 0$$

K vyřešení této rovnice použijme Lagrangeovy trigonometrické identity[zdroj], které vypadají následovně:

$$\sum_{n=1}^{N} \sin\left(n \theta \right) = \frac{1}{2} \cot\left(\frac{\theta}{2} \right) - \frac{\cos\left(\left(N + \frac{1}{2} \right) \theta \right)}{ 2 \sin\left(\frac{\theta}{2} \right) } \\ \sum_{n=1}^{N} \cos\left(n \theta \right) = -\frac{1}{2} + \frac{\sin\left(\left(N + \frac{1}{2} \right) \theta \right)}{ 2 \sin\left(\frac{\theta}{2} \right) }$$

Pro jejich použití je nutné si uvědomit, že v našem případě k = 0 je ekvivalentní s k = N, díky periodicitě. Tudíž můžu psát:

$$\sum_{k=1}^{N} \sin\left(k \frac{2\pi\left(n - m \right)}{N} \right) = \frac{1}{2} \cot\left(\frac{\pi\left(n - m \right)}{N} \right) - \frac{\cos\left(\left(N + \frac{1}{2} \right) \frac{2\pi\left(n - m \right)}{N} \right)}{ 2 \sin\left(\frac{\pi\left(n - m \right)}{N} \right) } =\\= \frac{1}{2} \cot\left(\frac{\pi\left(n - m \right)}{N} \right) - \frac{\cos\left(2\pi\left(n - m \right) + \frac{\pi\left(n - m \right)}{N} \right)}{ 2 \sin\left(\frac{\pi\left(n - m \right)}{N} \right) } = \frac{1}{2} \cot\left(\frac{\pi\left(n - m \right)}{N} \right) - \frac{1}{2} \cot\left(\frac{\pi\left(n - m \right)}{N} \right) = 0$$

A ekvivalentně pro reálnou část:

$$\sum_{k=1}^{N} \cos\left(k \frac{2\pi\left(n - m \right)}{N} \right) = -\frac{1}{2} + \frac{\sin\left(\left(N + \frac{1}{2} \right) \frac{2\pi\left(n - m \right)}{N} \right)}{ 2 \sin\left(\frac{\pi\left(n - m \right)}{N} \right) } =\\= -\frac{1}{2} + \frac{\sin\left(2\pi\left(n - m \right) + \frac{\pi\left(n - m \right)}{N} \right)}{ 2 \sin\left(\frac{\pi\left(n - m \right)}{N} \right) } = -\frac{1}{2} + \frac{1}{2} = 0$$

```
def xn ( frequency, N, n ):
    sum = 0
    for k in range( N ):
        sum += frequency[ k ] * cmath.exp( 1j * 2 * cmath.pi
* k * n / N )
    return sum / N

def idft ( frequency ):
    N = len( frequency )
    original = []
    for n in range( N ):
        original.append( xn( frequency, N, n ) )
    return original

orig = idft( dft( [9,7,9,2] ) )
printListOfComplexNumber( orig )
```

```
    9.00 + -0.00 j
    7.00 + -0.00 j
    9.00 + 0.00 j
    2.00 + 0.00 j
```

Rychlá Fourierova transformace (Fast Fourier Transform)

Diskrétní Fourierova transformace dle definice vyžaduje O\left(n^2\right) násobení. Pro její zrychlení se využívá algoritmus rychlé Fourierovy transformace. Nejznámější a nejčastěji používaný je **Cooley-Tukey**

algoritmus. Jehož kód pro počet prvků N = 2^k
zdroj]:

```
def fft ( original ):
    N = len( original )
    if ( N == 1 ):
        return original

    S = fft( original[0:N:2] )
    L = fft( original[1:N:2] )

    result = []
    for k in range( N/2 ):
        tmp = L[k] * cmath.exp( -2j * cmath.pi * k / N )
        result.insert( k, S[k] + tmp )
        result.insert( k + N/2, S[k] - tmp )

    return result


printListOfComplexNumber( fft( [ 9, 7, 9, 2 ] ) )
```

```
    27.00 + 0.00 j
    0.00 + -5.00 j
    9.00 + 0.00 j
    -0.00 + 5.00 j
```

Inverzní rychlá Fourierova tranformace je opět téměř stejná.

```
def ifft ( original ):
    N = len( original )
    if ( N == 1 ):
        return original

    S = fft( original[0:N:2] )
    L = fft( original[1:N:2] )

    result = []
    for k in range( N/2 ):
        tmp = L[k] * cmath.exp( 2j * cmath.pi * k / N )
        result.insert( k, S[k] + tmp )
        result.insert( k + N/2, S[k] - tmp )

    return [ n / N for n in result ]

printListOfComplexNumber( ifft( fft( [ 9, 7, 9, 2 ] ) ) )
```

```
    9.00 + 0.00 j
    7.00 + 0.00 j
    9.00 + 0.00 j
```

```
    2.00 + 0.00 j
```

Výhoda tohoto algoritmu je, že potřebuje jen $O\left(n\log\left(n\right)\right)$ násobení.

Obě dvě implementace jsou pouze pro $N = 2^k$
obecný zápis algoritmu, který zde neuvádíme.

Použití FFT

FFT se používá například pro násobení velkých čísel či polynomů. Klasické násobení polynomů má časovou složitost $O\left(n^2\right)$
jsme schopni tuto složitost schopni snížit na $O\left(n\log\left(n\right)\right)$
Pro násobení polynomů (a tedy i velkých čísel) platí, že součin polynomů je ekvivalentní součinu jednotlivých "frekvencí".

Například pro $\left(3x + 2\right) \cdot \left(x^2 + 7x - 8 \right)$
výsledek $3x^3 + 23x^2 - 10x - 16$
rozšířit tak, aby jejich řád byl $n = 2^k - 1 \wedge n >=$
největším řádem (výsledku).

```
def multiply ( l1, l2 ):
    res = []
    for i in range( len( l1 ) ):
        res.append( l1[ i ] * l2[ i ] )
    return res

polynom1 = [ 2, 3, 0, 0 ]
polynom2 = [ -8, 7, 1, 0 ]
vysledek = ifft( multiply( fft( polynom1 ), fft( polynom2 ) )
)

printListOfComplexNumber( vysledek )
```

```
    -16.00 + -0.00 j
    -10.00 + 0.00 j
    23.00 + 0.00 j
    3.00 + -0.00 j
```