Ivanov_lab7

March 24, 2020

Глава 7 (Иванов Илья, группа 3530901/70203)

1 Теория

Зная дискретное косинусное преобразование (ДКП), легко понять и ДПФ. Вся разница в том, что вместо косинусов используются комплексные экспоненциальные функции.

1.1 Комплексные экспоненты

Леонард Эйлер нашёл одно из самых полезных обобщений в прикладной математике - комплексную экспоненциальную функцию.

Простое определение потенцирования - это последовательное умножение, например $\phi^3 = \phi \cdot \phi \cdot \phi$. Но это определение не годится для дробных степеней.

Потенцирование можно выразить в виде степенного ряда:

$$e^{\phi} = 1 + \phi + \phi^2/2! + \phi^3/3! + ...$$

Это определение верно для вещественных чисел, для мнимых чисел и, простым расширением, для комплексных чисел. Применив это определение к чисто мнимому числу $i\phi$, получим:

$$e^{i\phi} = 1 + i\phi - \phi^2/2! - i\phi^3/3! + ...$$

Перестановкой членов можно показать, что это эквивалентно

$$e^{i\phi} = \cos\phi + i\sin\phi.$$

Предполагается, что $e^{i\phi}$ - это комплексное число с модулем 1; соответствующая ему точка на комплексной плоскости всегда расположена на единичной окружности. Если представить число как вектор, то угол ϕ между вектором и положительной осью x, выраженный в радианах, - это аргумент.

В случае, когда показатель степени - комплексное число, получим:

$$e^{\alpha + i\phi} = e^{\alpha}e^{i\phi} = Ae^{i\phi}$$

где A - действительное число, определяющее модуль (амплитуду), а $e^{i\phi}$ - единичное комплексное число, определяющее угол (фазу).

1.2 Комплексные сигналы

Если $\phi(t)$ функция времени, $e^{i\phi(t)}$ - также функция времени. В частности:

$$e^{i\phi(t)} = \cos\phi(t) + i\sin\phi(t)$$

Эта функция описывает величину, изменяющуюся во времени, то есть сигнал. В частности, это комплексный экспоненциальный сигнал.

В особом случае, когда частота сигнала постоянна, $\phi(t)$ есть $2\pi ft$, а результат - комплексная синусоида:

$$e^{i2\pi ft} = \cos 2\pi ft + i\sin 2\pi ft$$

В общем случае у сигнала может быть ненулевая начальная фаза ϕ_0, что даёт:

$$e^{i(2\pi ft+\phi_0)}$$

1.3 Задача синтеза

Сложные сигналы можно создавать сложением не только действительных, но и комплексных синусоид с разными частотами. Это задача синтеза в комплексной форме: как оценить сигнал, имея частоты и амплитуды каждой комплексной компоненты?

Простейшее решение - создать объекты `ComplexSinusoid` и сложить их.

Попросту говоря, комплексный сигнал - это последовательность комплексных чисел. Но как его интерпретировать? С действительными сигналами всё понятно: величины, изменяющиеся во времени. Но обычные измерения никогда не дают комплексных чисел.

Так что же это такое - комплексный сигнал? Лучшее, что можно предложить, - два частных ответа:

1) Комплексный сигнал есть математическая абстракция, полезная при расчётах и анализе, но напрямую она не соответствует ничему в реальном мире.

2) Комплексный сигнал - это последовательность комплексных чисел, то есть два сигнала, составляющие действительную и мнимую части.

1.4 Задача анализа

Задача анализа обратна задаче синтеза: имея последовательность образцов, y , и зная частоты, имеющиеся в сигнале, как вычислить комплексные амплитуды компонент?

В главе №6 было показано, как можно решить эту проблему созданием матрицы синтеза M и решением системы линейных уравнений $M\alpha = y$.

1.5 Эффективный анализ

К сожалению, решение системы линейных уравнений идёт медленно. ДКП получилось ускорить таким выбором `fs` и `ts`, чтобы M стала ортогональной. Тогда обратная M аналогична транспонированной M; значит, и ДКП, и обратное ДКП можно вычислить перемножением матриц.

Для ДПФ всё работает аналогично, но с небольшим изменением. Так как M комплексная, она должна быть унитарной, а не ортогональной, так что обратная M будет комплексно-сопряжённой к транспонированной M, которую можно вычислить транспонированием матрицы и переменой знака у мнимой части каждого элемента.

1.6 Периодичность ДПФ

В этой главе ДПФ представлено в виде перемножения матриц. Вычисляется матрица синтеза M и матрица анализа $M*$. При умножении $M*$ на массив сигнала y каждый элемент результата есть произведение строки из $M*$ и y, которое можно записать в виде суммы:

$$DFT(y)[k] = \sum_n y[n] exp(-2\pi ink/N),$$

где k - индекс частоты от 0 до $N-1$, а n - индекс времени от 0 до $N-1$. Так что $DFT(y)[k]$ - это k-й элемент ДПФ от y.

Обычно это сумма N значений k, в порядке от 0 до $N-1$. Можно оценить её и для иных значений k, но в этом нет смысла, так как они начинают повторяться. Иными словами, значение в k то же, что и в $k+N$ или $k+2N$ или $k-N$ и т.д. ДПФ периодично, с периодом N.

2 Упражнения

[1]:
```
from __future__ import print_function, division

import thinkdsp
import thinkplot

import numpy as np

import warnings
warnings.filterwarnings('ignore')

PI2 = 2 * np.pi

np.set_printoptions(precision=3, suppress=True)
%matplotlib inline
```

2.1 Упражнение 7.1.

The notebook for this chapter, chap07.ipynb, contains additional examples and explanations. Read through it and run the code.

`chap07.ipynb` был просмотрен и разобран.

2.2 Упражнение 7.2.

In this chapter, I showed how we can express the DFT and inverse DFT as matrix multiplications. These operations take time proportional to N^2, where N is the length of the wave array. That is fast enough for many applications, but there is a faster algorithm, the Fast Fourier Transform (FFT), which takes time proportional to $N \log N$.

The key to the FFT is the Danielson-Lanczos lemma:

$$DFT(y)[n] = DFT(e)[n] + \exp(-2\pi in/N)DFT(o)[n]$$

Where $DFT(y)[n]$ is the nth element of the DFT of y; e is a wave array containing the even elements of y, and o contains the odd elements of y.

This lemma suggests a recursive algorithm for the DFT:1. Given a wave array, y, split it into its even elements, e, and its odd elements, o. 2. Compute the DFT of e and o by making recursive calls. 3. Compute DFT(y) for each value of n using the Danielson-Lanczos lemma.

For the base case of this recursion, you could wait until the length of y is 1. In that case, $DFT(y) = y$. Or if the length of y is sufficiently small, you couldcompute its DFT by matrix multiplication, possibly using a precomputed matrix.

Hint: I suggest you implement this algorithm incrementally by starting with a version that is not truly recursive. In Step 2, instead of making a recursive call, use dft, as defined by Section 7.7, or np.fft.fft. Get Step 3 working, and confirm that the results are consistent with the other implementations. Then add a base case and confirm that it works. Finally, replace Step 2 with recursive calls.

One more hint: Remember that the DFT is periodic; you might find np.tile useful.

Начнём с небольшого реального сигнала и вычислим его БПФ.

[2]:
```
ys = [-0.5, 0.1, 0.7, -0.1]
hs = np.fft.fft(ys)
print(hs)
```

```
[ 0.2+0.j  -1.2-0.2j  0.2+0.j  -1.2+0.2j]
```

Класс, реализующий ДПФ:

[3]:
```
def dft(ys):
    N = len(ys)
    ts = np.arange(N) / N
    freqs = np.arange(N)
    args = np.outer(ts, freqs)
    M = np.exp(1j * PI2 * args)
    amps = M.conj().transpose().dot(ys)
    return amps
```

Проверим, выдаёт ли эта реализация тот же результат, что и `np.fft.fft`:

[4]:
```
hs2 = dft(ys)
print(sum(abs(hs - hs2)))
```

```
5.864775846765962e-16
```

Результаты одинаковые с точностью до ошибок округления.

В качестве шага к созданию рекурсивного БПФ начнём с версии, которая разбивает входной массив на две части и использует `np.fft.fft` для вычисления БПФ половин:

[5]:
```
def fft_norec(ys):
    N = len(ys)
    He = np.fft.fft(ys[::2])
    Ho = np.fft.fft(ys[1::2])

    ns = np.arange(N)
    W = np.exp(-1j * PI2 * ns / N)

    return np.tile(He, 2) + W * np.tile(Ho, 2)
```

Получаем такой же результат:

[6]:
```
hs3 = fft_norec(ys)
print(sum(abs(hs - hs3)))
```

```
0.0
```

Наконец, заменим `np.fft.fft` рекурсивными вызовами и добавим базовый случай:

[7]:
```
def fft(ys):
    N = len(ys)
    if N == 1:
        return ys

    He = fft(ys[::2])
    Ho = fft(ys[1::2])

    ns = np.arange(N)
    W = np.exp(-1j * PI2 * ns / N)

    return np.tile(He, 2) + W * np.tile(Ho, 2)
```

Опять получаем тот же результат:

[8]:
```
hs4 = fft(ys)
print(sum(abs(hs - hs4)))
```

```
1.6653345369377348e-16
```

Эта реализация FFT занимает время, пропорциональное $N \log N$ и память, пропорциональную $N \log N$, а также тратит время на создание и копирование массивов.

3 Вывод

В ходе выполнения данной лабораторной работы было рассмотрено дискретное преобразование Фурье и быстрое преобразование Фурье и их практическое применение.