Лабораторная работа №1Быстрое преобразование Фурье

Выполнил Казачинский Глеб **3** курс **6** группа

Постановка задачи

Написать программу, которая реализует функцию быстрого преобразования Фурье `fft`, а также обратную к ней `ifft`, указанным в варианте способом. Язык программирования может быть любым, но должен иметь библиотеку или встроенные функции для вычисления быстрого преобразования Фурье (эту функцию обозначим `truefft` и будем использовать для проверки результатов).

Провести вычислительные эксперименты по следующей схеме. Для $n = 2^k, k = 1, 2, \ldots, 16$ сгенерировать случайный вектор $x^n \in \mathbb{C}^n$ и вычислить для него векторы y^n=`fft`(x^n), $\tilde{x}^n$=`ifft`(y^n) и z^n=`truefft`(x^n). Вычислить $\epsilon^n = \|x^n - \tilde{x}^n\|$ и $\delta^n = \|y^n - z^n\|$. Время вычисления вектора y^n обозначим t_y^n, время вычисления вектора z^n обозначим t_z^n.

Требования к содержанию отчета

1) Векторы $x^8, \tilde{x}^8, y^8, z^8, \epsilon^8, \delta^8$.
2) Точечные графики t_y^n и t_z^n (на одной координатной плоскости).
3) Таблицу, каждая строчка которой содержит k, ϵ^n, δ^n, t_y^n, t_z^n.
4) Ваши комментарии и выводы.
5) Исходный код программы.

Отчет о работе печатается на принтере. Титульный лист не нужен! В заголовке работы указывается номер и тема лабораторной работы, имя и фамилия автора, а также скриншот требований к содержанию из настоящего документа. Листы отчета должны быть скреплены!

Вместе с отчетом на защиту лабораторной работы приносится ноутбук с разработанной программой, которую нужно будет выполнить.

Вариант 2

Реализовать алгоритм БПФ на основе факторизации Кули-Тьюки. Перестановку P_n^T рекомендуется вычислять с помощью бит-реверсии.

Код Программы:

```
import numpy as np
import scipy.linalg as scpl
import time
import pandas as pd
from matplotlib import pyplot as plt


def get_degree_of_2(n):
    degree = 0
    while n >= 2:
        n /= 2
        degree += 1
    return degree


def w(n, m):
    return np.exp((-2 * m * np.pi / n) * 1j)


def w_inv(n, m):
    return np.exp((2 * m * np.pi / n) * 1j)


def fft(x, is_inverse=False):
    t = get_degree_of_2(len(x))
    y = np.array(x[bit_reverse(len(x))], dtype=complex)
    for q in range(t):
        y = mul_A_B(y, q + 1, is_inverse)
    if is_inverse:
        y /= len(y)
    return y

```

```python

def truefft(x):
    return np.fft.fft(x)


def mul_A_B(y, q, is_inverse):
    L = 2 ** q
    y1 = y.copy()
    for i in range(0, len(y), L):
        y2 = y[i:i + L]
        y3 = y2.copy()
        if not is_inverse:
            for j in range(len(y2) // 2):
                y3[j] = y2[j] + w(len(y2), j) * complex(y2[len(y2) // 2 + j])
            for j in range(len(y2) // 2, len(y2)):
                y3[j] = y2[j - len(y2) // 2] - w(len(y2), j - len(y2) // 2) * complex(y2[j])
        else:
            for j in range(len(y2) // 2):
                y3[j] = y2[j] + w_inv(len(y2), j) * complex(y2[len(y2) // 2 + j])
            for j in range(len(y2) // 2, len(y2)):
                y3[j] = y2[j - len(y2) // 2] - w_inv(len(y2), j - len(y2) // 2) * complex(y2[j])
        y1[i:i + L] = y3
    return y1


def bit_reverse(n):
    reshuffle = []
    bit_length = get_degree_of_2(n)
    for num in range(n):
        size = bit_length - 1
        reversed_num = 0
```

```
        while num > 0:
            k = num % 2
            num //= 2
            reversed_num += k << size
            size -= 1
        reshuffle.append(reversed_num)
    return reshuffle


t_y = []
t_z = []
eps_arr = []
delta_arr = []
for k in range(16):
    n = 2 ** k
    x_n = (np.random.rand(n) + np.random.rand(n) * 1j)

    start_time = time.time()
    y_n = fft(x_n)
    t_y.append((time.time() - start_time) * 1000)

    x_n_inv = fft(y_n, True)

    start_time = time.time()
    z_n = truefft(x_n)
    t_z.append((time.time() - start_time) * 1000)

    eps = scpl.norm(x_n - x_n_inv)
    delta = scpl.norm(y_n - z_n)
    delta_arr.append(delta)
    eps_arr.append(eps)
    if n == 8:
        print('x_8:', x_n)
```

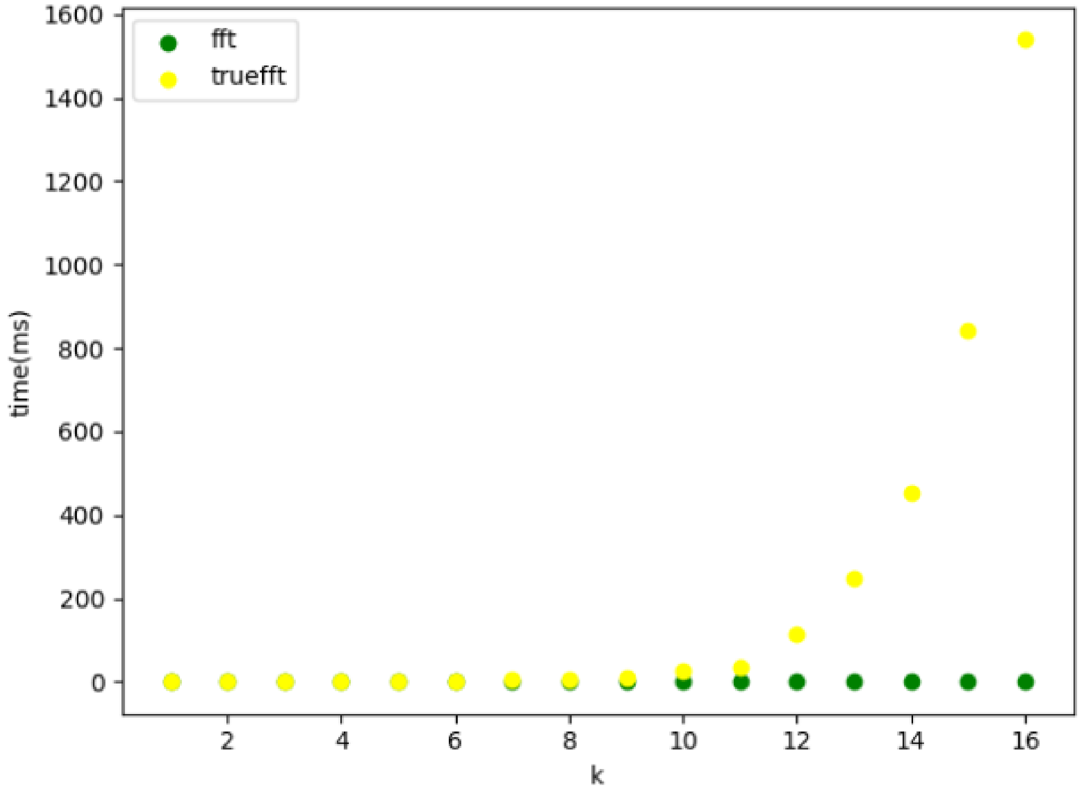
```python
        print('x_8_inv:', x_n_inv)
        print('y_8:', y_n)
        print('z_8:', z_n)
        print('eps_8:', eps)
        print('delta_8:', delta)

plt.figure()
plt.scatter(range(1, 17), t_z, color='green')
plt.scatter(range(1, 17), t_y, color='yellow')
plt.xlabel('k')
plt.ylabel('time(ms)')
plt.legend(['fft', 'truefft'])
plt.show()

print(pd.DataFrame(dict(k=range(1, 17), t_y=t_y, t_z=t_z, eps=eps_arr,
delta=delta_arr)))
```

Результаты работы программы:

1) Векторы $x^8, \tilde{x}^8, y^8, z^8, \epsilon^8, \delta^8$.

```
x_8: [0.41701073+0.30094098j 0.64099182+0.37766592j 0.83745534+0.9432493j
 0.57653692+0.60106964j 0.2334906 +0.47733405j 0.61446448+0.28791253j
 0.37311152+0.78911632j 0.02752292+0.45307469j]
x_8_inv: [0.41701073+0.30094098j 0.64099182+0.37766592j 0.83745534+0.9432493j
 0.57653692+0.60106964j 0.2334906 +0.47733405j 0.61446448+0.28791253j
 0.37311152+0.78911632j 0.02752292+0.45307469j]
y_8: [ 3.72058434e+00+4.23036342j  1.36312712e-01-1.08888908j
 -9.48631398e-01-1.60548705j  5.66954479e-01-0.07783544j
  1.55204258e-03+0.79091788j  5.38993495e-01-0.19258471j
 -1.71499652e-01-0.30269413j -5.08180168e-01+0.65373692j]
z_8: [ 3.72058434e+00+4.23036342j  1.36312712e-01-1.08888908j
 -9.48631398e-01-1.60548705j  5.66954479e-01-0.07783544j
  1.55204258e-03+0.79091788j  5.38993495e-01-0.19258471j
 -1.71499652e-01-0.30269413j -5.08180168e-01+0.65373692j]
eps_8: 1.9229626863835638e-16
delta_8: 2.7336071744532853e-16
```

2) Точечные графики t_u^n и t_z^n (на одной координатной плоскости).



3) Таблицу, каждая строчка которой содержит k, ϵ^n, δ^n, t_y^n, t_z^n.

k	t_y	t_z	eps	delta
1	0.019073	0.028849	0.000000e+00	0.000000e+00
2	0.049829	0.020981	1.110223e-16	0.000000e+00
3	0.051022	0.011921	1.841750e-16	7.850462e-17
4	0.107050	0.010967	2.573946e-16	3.421937e-16
5	0.388145	0.037909	4.340554e-16	1.034058e-15
6	2.177000	0.098705	7.176306e-16	2.413937e-15
7	1.853943	0.082731	1.238238e-15	6.556664e-15
8	3.478050	0.027895	2.071606e-15	1.515556e-14
9	6.770134	0.026941	3.440243e-15	3.773725e-14
10	17.652988	0.525951	4.950817e-15	8.445321e-14
11	61.880112	0.313997	6.961757e-15	1.894576e-13
12	100.762129	0.077009	1.056086e-14	4.055482e-13
13	152.705193	0.132084	1.599991e-14	8.731920e-13
14	345.734835	0.245094	2.316968e-14	1.871853e-12
15	755.976915	1.974106	3.445027e-14	3.967432e-12
16	1824.460030	1.012802	5.053138e-14	8.545682e-12

4) Ваши комментарии и выводы.

Нормы векторов ошибки имеют порядок 10 − 16. Можно утверждать, что алгоритм работает корректно.

В таблице и на графике зависимости t от k видим, что реализованный алгоритм работает существенно медленнее встроенного fft.

Итого, в данной работе был реализован алгоритм быстрого преобразования Фурье на основе факторизации Кули-Тьюки и алгоритм обратного преобразования. Проведена проверка корректности реализованного алгоритма путём сравнения с втроенным fft.