实验二：快速傅里叶变换

胡子畅　23336091

3. 使用快速傅里叶变换确定函数 $f(x)=x^2\cos x$ 在$[-\pi,\pi]$上的 16 次三角插值多项式.

一、快速傅里叶变换的原理

快速傅里叶变换(FFT)是20世纪最重要的算法之一，它将离散傅里叶变换(DFT)的计算复杂度从$\mathcal{O}(n^2)$降低到$\mathcal{O}(n\log n)$。让我们一步步探索这个算法背后的优美数学直觉。

1. 多项式表示法的转换

首先，我们考虑多项式的两种表示方式：

- **系数表示法**：对于一个$n-1$次多项式

$$P(x)=a_0+a_1x+a_2x^2+\cdots+a_{n-1}x^{n-1}$$

我们可以用系数向量$(a_0,a_1,\ldots,a_{n-1})$表示。

- **点值表示法**：选择n个不同的点$x_0,x_1,\ldots,x_{n-1}$，计算$y_k=P(x_k)$，则$(x_k,y_k)_{k=0}^{n-1}$构成点值表示。

关键思想：在特定点集上，这两种表示法可以高效转换。FFT的核心就是找到这样的点集和转换方法。

2. 正负对的巧妙利用

假设我们想计算多项式$P(x)$在
$x = \pm 1, \pm 2, \ldots, \pm n/2$处的值。观察：

对于偶函数部分$P_e(x) = \frac{P(x)+P(-x)}{2}$和奇函数部分$P_o(x) = \frac{P(x)-P(-x)}{2}$，有：

$$\begin{cases} P(x) = P_e(x) + xP_o(x) \\ P(-x) = P_e(x) - xP_o(x) \end{cases}$$

这样只需要计算P_e和P_o在正点处的值，就能得到正负两点的值。这使计算量减半！

3. 引入复数与单位根

但仅用实数点无法递归应用上述思想。**天才的突破**：使用复数，特别是1的n次单位根$\omega_n = e^{2\pi i/n}$。

单位根有以下优美性质：

$$\begin{aligned} \omega_n^n &= 1 \\ \omega_n^{k+n/2} &= -\omega_n^k \quad \text{（正负对）} \\ \omega_n^{2k} &= \omega_{n/2}^k \quad \text{（规模减半）} \end{aligned}$$

选择点集为ω_n^k（$k = 0, 1, \ldots, n-1$），我们可以递归地将问题分解。

4. FFT算法

对于多项式$P(x) = \sum_{k=0}^{n-1} a_k x^k$，在单位根处的值为：

$$P(\omega_n^j) = \sum_{k=0}^{n-1} a_k \omega_n^{jk}$$

将系数分为偶数下标和奇数下标：

$$P(x) = P_e(x^2) + xP_o(x^2)$$

利用单位根性质$\omega_n^{j+n/2} = -\omega_n^j$，我们得到FFT的分治算法：

1. 计算P_e和P_o在$\omega_{n/2}^j$（$j = 0, \ldots, n/2-1$）处的值
2. 合并结果：

$$\begin{cases} P(\omega_n^j) = P_e(\omega_{n/2}^j) + \omega_n^j P_o(\omega_{n/2}^j) \\ P(\omega_n^{j+n/2}) = P_e(\omega_{n/2}^j) - \omega_n^j P_o(\omega_{n/2}^j) \end{cases}$$

5. 逆FFT算法

逆变换与正变换形式几乎相同：

$$a_k = \frac{1}{n} \sum_{j=0}^{n-1} P(\omega_n^j) \omega_n^{-jk}$$

区别仅在于：

1. 使用ω_n^{-1}代替ω_n
2. 最后结果要除以n

这种对称性使得同一套算法框架可以处理正逆变换。

6. 编程实现

```
def fft(a, invert=False):
    n = len(a)
    if n == 1:
        return a
    a0 = fft(a[::2], invert)
    a1 = fft(a[1::2], invert)
    ang = 2 * pi / n * (-1 if invert else 1)
    w, wn = 1, complex(cos(ang), sin(ang))
    a = [0] * n
    for i in range(n//2):
        a[i] = a0[i] + w * a1[i]
        a[i + n//2] = a0[i] - w * a1[i]
        if invert:
            a[i] /= 2
            a[i + n//2] /= 2
        w *= wn
    return a
```

二、快速傅里叶变换的应用

1. 采样策略

对于周期为2π的函数$f(x)$，我们在区间$[-\pi, \pi]$上进行均匀采样：

$$x_j = -\pi + \frac{2\pi j}{N}, \quad j = 0, 1, \ldots, N-1$$

其中采样点数N必须为2的幂次。这种采样方式保证了：

$$e^{ikx_j} = e^{ik(-\pi + \frac{2\pi j}{N})}$$

具有完美的周期性特征。

2. 傅里叶系数的离散计算

通过离散傅里叶变换，傅里叶系数可表示为：

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j) e^{-ikx_j}, \quad k = 0, 1, \ldots, N-1$$

3. 频率分量的对称处理

在频率域中，系数需要重新排列：

- 正频率部分：$k = 0, \ldots, N/2 - 1$
- 负频率部分：$k = N/2, \ldots, N-1$对应实际频率$k - N$

对于实值函数，系数满足共轭对称性：

$$c_{N-k} = c_k^*$$

4. 插值多项式的构造

最终的三角插值多项式为：

$$p(x) = \sum_{k=-N/2}^{N/2-1} c_k e^{ikx} = \sum_{k=0}^{N/2-1} c_k e^{ikx} + \sum_{k=N/2}^{N-1} c_k e^{i(k-N)x}$$

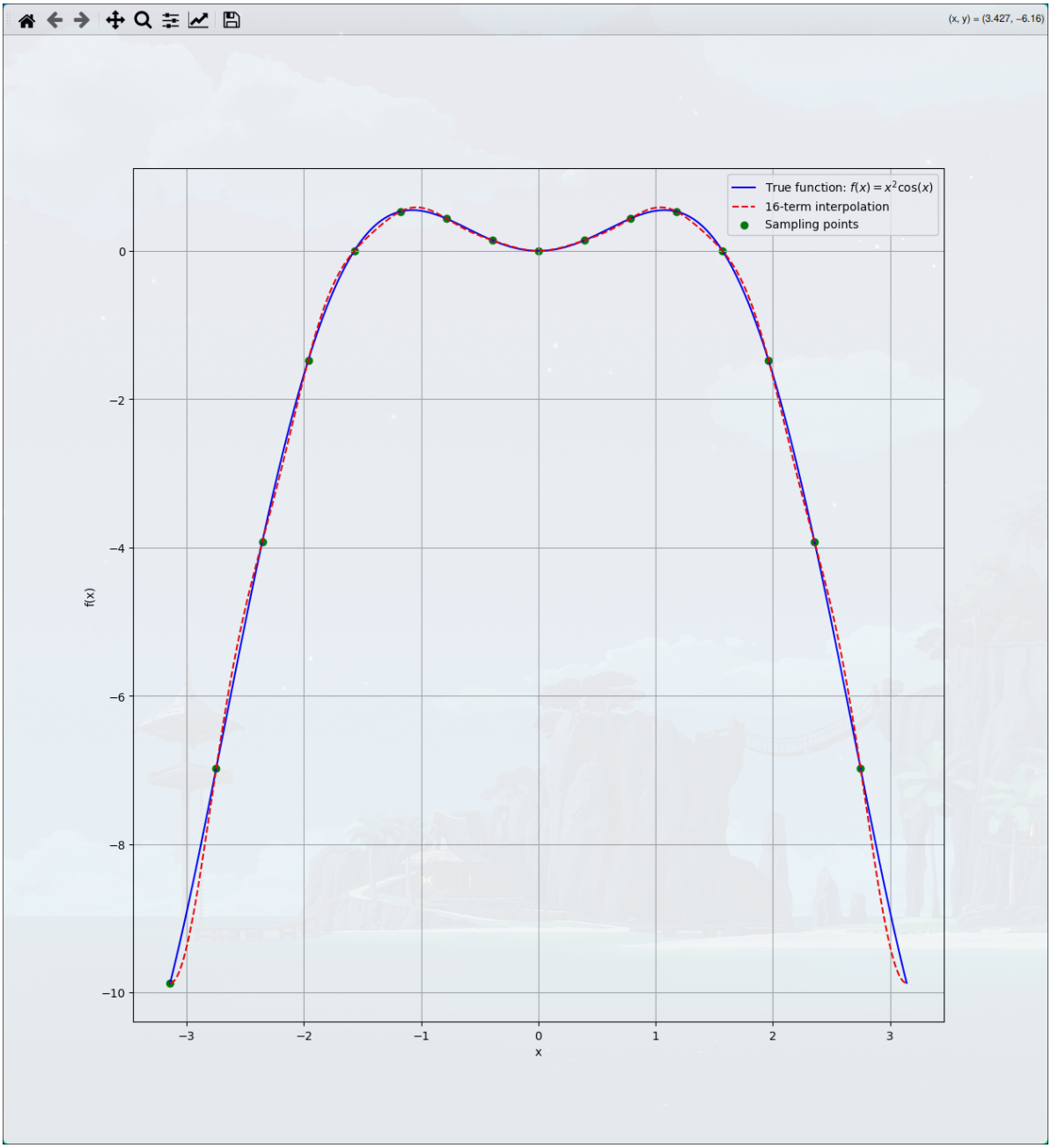
5. 编程实现

```
# 待插值函数
def f(x):
    return x**2 * np.cos(x)

# 参数设置
N = 16          # 采样点数（需为 2 的幂）

# 采样
x_samples = np.linspace(-pi, pi, N,
endpoint=False)
y_samples = f(x_samples)

# 利用IFFT计算插值多项式的系数
coeffs = fft(y_samples, invert=True)

# 三角插值多项式
def p(x):
    result = np.zeros_like(x, dtype=complex)
    # 将[-pi,pi]映射到[0,2*pi]
```

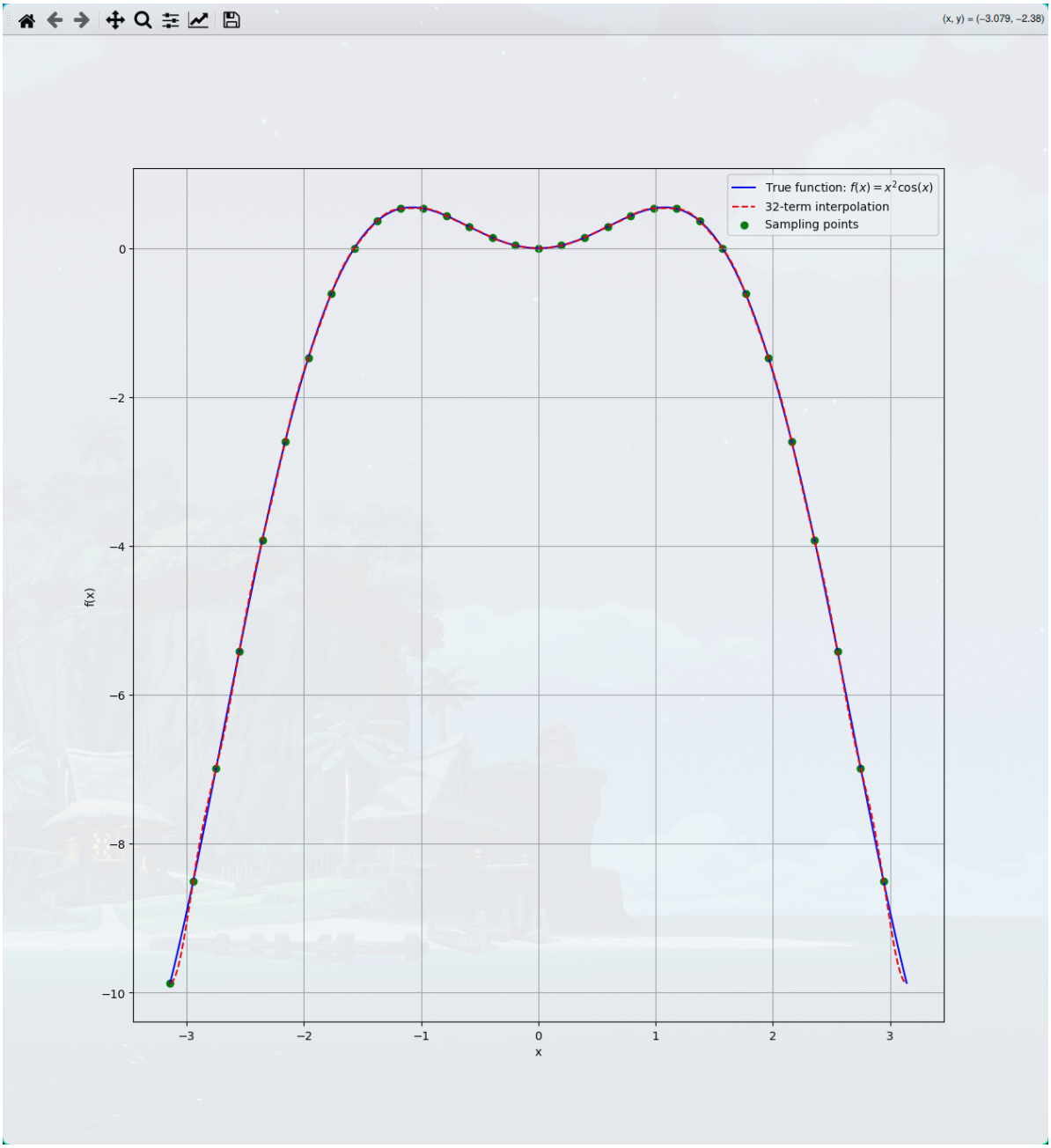
```
    x = (x + pi) % (2 * pi)
    # 方法一：正负频率分开计算
    # 正频率分量
    for k in range(N//2):
        result += coeffs[k] * np.exp(1j * k
* x)
    # 负频率分量
    for k in range(N//2, N):
        result += coeffs[k] * np.exp(1j * (k
- N) * x)
    # 方法二：正负频率合一计算（利用python负索引的
特性）
    for k in range(-N//2, N//2):
        result += coeffs[k] * np.exp(1j * k
* x)
    return result.real
```

6. 图像生成

采样点数为16时：


(x, y) = (3.427, −6.16)
True function: $f(x) = x^2\cos(x)$
16-term interpolation
Sampling points
f(x)
x
0
−2
−4
−6
−8
−10
−3
−2
−1
0
1
2
3

采样点数为32时：


(x, y) = (−3.079, −2.38)
True function: $f(x) = x^2\cos(x)$
32-term interpolation
Sampling points
f(x)
x
0
−2
−4
−6
−8
−10
−3
−2
−1
0
1
2
3