编程作业三报告

刘翰文

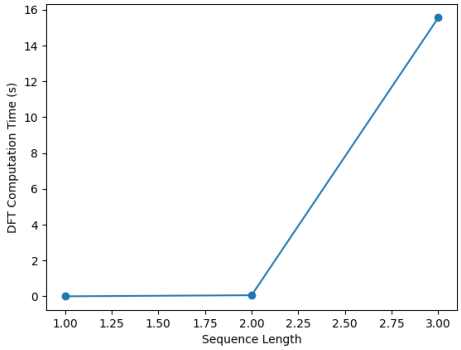
522030910109

本次作业使用 python 完成，作业要求分别按照 DFT 定义、使用三种 FFT 实现对一个序列的 DFT 计算，并记录不同序列长度的计算时间并画图。

首先按照定义写出 DFT：

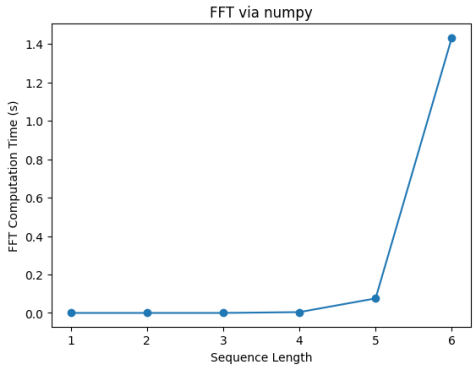
```python
def DFT(x):
    N = len(x)
    X = np.zeros(N,dtype='complex_')
    for k in range(N):
        for n in range(N):
            X[k] = X[k] + x[n]*np.exp(-1j*2*np.pi*n*k/N)
```

运行 DFT 对于一定长度的序列，得到结果(横轴坐标 x 代表序列长度为2^{4x})：



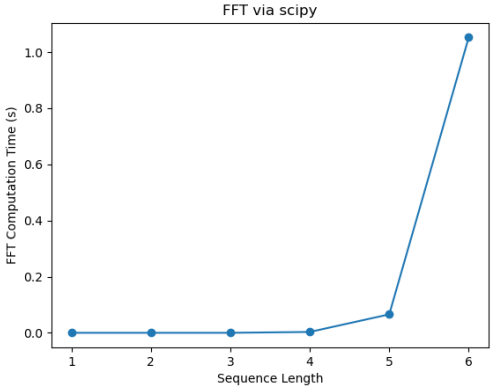
x=4 时，代码运行时间过长无法跑完，故只展示 x=1,2,3。可以看到 x=1,2 时，运行时间接近 0，而 x 一旦到 3，则运行时间急剧上升，这也与 DFT 的运行时间为$O(n^2)$和 python 本身运行速度过慢有关。

接下来，首先使用 numpy 库中的 fft 函数求解序列的 DFT，得到的结果如下(x=7 时，内存不够)：



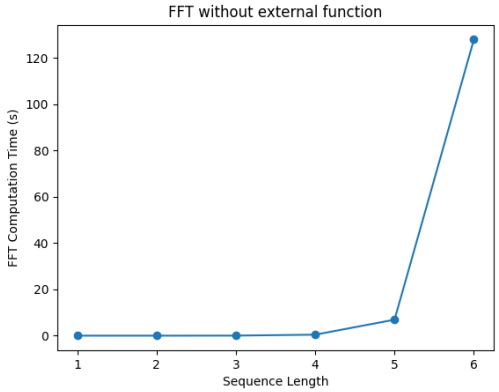
从结果来看，numpy 中 fft 的运行时间远远小于 DFT 直接计算结果，一部分原因是由于 fft 算法带来的优化，从$O(n^2)$到$O(nlogn)$。另一部分原因是由于 numpy 库函数本身的实现方法，极大地提高了代码运行速度。

接下来，使用 scipy 库中的 fft 函数求解序列的 DFT，得到的结果如下(x=7 时，内存不够)：



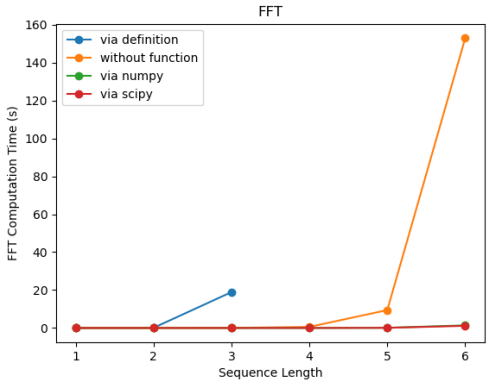
结果与 numpy 库中的 fft 函数得出的结果相似，计算时间显著短与 DFT 直接计算的结果。

最后，不使用库中 fft 函数，直接编写 fft 代码求解序列的 DFT，得到的结果如下(x=7 时，内存不够)：



从结果图看出，其运行时间与库函数相比，运行时间要显著的长，但与 DFT 定义求解相比，仍然显著要快。

将三种 FFT 和 DFT 结果结合在一起，得到：



从这张结果图也可以得到整个问题的结论：FFT 极大加速了 DFT 的计算，且不同 FFT 的实现方法对速度也有很大影响，numpy 和 scipy 的 fft 函数效率要远高于手动实现的 fft 函数。